

```

/*****
/*                                W I N . C                                */
/**-----**/
/* task          : control text mode window                                */
/**-----**/
/* author         : Michael Tischer / Bruno Jennrich                      */
/* developed on    : 5/3/1994                                              */
/* last update     : 4/07/1995                                            */
/**-----**/
/* COMPILER       : Borland C++ 3.1, Microsoft Visual C++ 1.5          */
/*****
#ifdef __WIN_C
/* WIN.C can also be #Included! */
#define __WIN_C

#include <dos.h>
#include <stdarg.h>
#include <stdio.h>
#include <conio.h>
#include <malloc.h>

#include "win.h"
#include "kbd.h"

/*- global variables -----*/
static LPBYTE _lpVIOSMEM = NULL; /* screen memory address */
static WORD _wVIOSMODE = 0;      /* actual video mode */
static INT _Columns = 0;         /* number of lines presented */

/*****
/* win_Init : initializes WINDOW structure                                */
/**-----**/
/* input : pWin - address of the WINDOW structure to be initialized */

```

```

/*          iX, iY - window position relative to subordinate          */
/*          originating point for the window, or screen : (0,0)      */
/*          iW, iH   - window width and height                       */
/*          iLA, iHA - text attributes (passive, active)             */
/*          uFlags - Flags                                           */
/*****
VOID _FP win_Init( PWINDOW pWin,
                  INT iX, INT iY,
                  INT iW, INT iH,
                  BYTE iLA, BYTE iHA,
                  UINT uFlags )
{
    pWin->iX      = iX;                      /* set parameter */
    pWin->iY      = iY;
    pWin->iW      = iW;
    pWin->iH      = iH;
    pWin->iLoAttr = iLA;
    pWin->iHiAttr = iHA;

    pWin->uFlags = uFlags;

    pWin->iCX = 0;                          /* cursor position = origin (source)of the window */
    pWin->iCY = 0;
    win_LoVideo( pWin );
}

/*****
/* win_LoVideo : switch print out to lowest intensity              */
/**-----**
/* input : pWin - window address, in which the following print outs */
/*          should be found having lower intensity.                  */
/*****

```

```

VOID _FP win_LoVideo( PWINDOW pWin )
{
    pWin ->iAttr = pWin->iLoAttr;
}

/*****
/* win_HiVideo : switch print out to high intensity          */
/**-----*/
/* input : pWin - window address, in which the following print outs */
/*          should be found with higher intensity.              */
/*****
VOID _FP win_HiVideo( PWINDOW pWin )
{
    pWin ->iAttr = pWin->iHiAttr;
}

/*****
/* win_GetVIOGMEM : transmits screen memory start address    */
/*                  and actual video mode.                    */
/**-----*/
/* Info : The screen memory address is saved in the          */
/*          global variables '_lpVIOGMEM'. The video mode is in */
/*          '_iVIOSMODE'.                                       */
/*****
VOID _FP win_GetVIOGMEM( VOID )
{
    static union _REGS regs;

    if( _lpVIOGMEM ) return; /* address already transmitted ? */

    regs.h.ah = VIOSETPAGE;
    regs.h.al = VIOPAGE;
    _int86( VIOINT, &regs, &regs );
}

```

```

regs.h.ah = VIOGETMODE;
regs.h.al = VIOPAGE;
_int86( VIOINT, &regs, &regs );
_Columns = regs.h.ah;                                /* number of columns */
_wVIOSMODE = regs.h.al;
if( _wVIOSMODE == 0x07 ) _lpVIOSMEM = MK_FP( 0xB000, 0 );
else                      _lpVIOSMEM = MK_FP( 0xB800, 0 );
}

/*****
/* win_GetMode : Transmits actual screen mode.                                */
/**-----**/
/* output      : VM_COLOR (1) - color screen                                */
/*              VM_MONO  (2) - monochrome screen                            */
/*              otherwise - unrecognized!                                    */
/*****
WORD _FP win_GetMode( VOID )
{
    win_GetVIOSMEM();
    return _wVIOSMODE;
}

/*****
/* win_GotoXY : set print out position within a window                        */
/**-----**/
/* input : pWin - window address, in which a new print out                    */
/*              position should be set.                                         */
/*              iX, iY - new position                                           */
/**-----**/
/* Info : If the flag "WIN_HASCURSORS" is set, the blinking                    */
/*              screen cursor is also set in this function.                    */

```

```

/*****
VOID _FP win_GotoXY( PWINDOW pWin, INT iX, INT iY )
{ static union _REGS regs;

    if( iX < 0 ) iX = 0;          /* test the validity of the new position */
    if( iY < 0 ) iY = 0;
    if( iX > pWin->iW ) iX = pWin->iW - 1;
    if( iY > pWin->iH ) iY = pWin->iH - 1;

    pWin->iCX = iX;
    pWin->iCY = iY;

    if( pWin->uFlags & WIN_ACTIVE )          /* window active ? */
    {
        if( pWin->uFlags & WIN_HASCURSOR ) /* Does the window have the cursor? */
        {
            regs.h.dh = ( BYTE ) ( pWin->iY + pWin->iCY );
            regs.h.dl = ( BYTE ) ( pWin->iX + pWin->iCX );
        }
        else
        {
            regs.h.dh = 0xFF;
            regs.h.dl = 0xFF;                /* No! */
        }

        regs.h.ah = VIOSETCURSOR;
        regs.h.bh = VIOPAGE;
        _int86( VIOINT, &regs, &regs );
    }
}

*****/

```

```

/* win_Clr : erase window contents */
/**-----**/
/* input : pWin - address of the window to be erased */
/******/
VOID _FP win_Clr( PWINDOW pWin )
{
    static INT iX, iY;
    static LPWORD lpRow;

    win_GetVIOSMEM();
    for( iY = 0; iY < pWin->iH; iY++ )
    {
        static WORD iAdr;
        iAdr = (pWin->iX+(iY+pWin->iY)*_Columns)*2;
        lpRow = ( LPWORD )&_lpVIOSMEM[ iAdr ];
        iAdr = MAKEWORD( pWin->iAttr, ' ' );
        for( iX = 0; iX < pWin->iW; iX++ ) *( lpRow++ ) = iAdr;
    }
    pWin->iCX = 0;
    pWin->iCY = 0;
}

/******/
/* win_Save : save window content */
/**-----**/
/* input : pWin - address of the window to be saved */
/* output : memory address, which recorded the window information */
/* together with the content, or ZERO if no other free */
/* memory is available. */
/******/
PBYTE _FP win_Save( PWINDOW pWin )
{
    PBYTE pMem;
    LPWORD lpRows;
    PWINDOW pWinData;

```

```

win_GetVIOSMEM();
pMem = malloc( ( pWin->iW * pWin->iH * 2 ) + sizeof( WINDOW ) );
if( pMem )
{
    static INT iY, iX;
    pWinData = ( PWINDOW )pMem;
    *pWinData = *pWin;
    lpRows = ( PWORD ) &pWinData[ 1 ];
    for( iY = 0; iY < pWin->iH; iY++ )
    {
        static WORD iAdr;
        static LPWORD lpR;
        iAdr = (pWin->iX+(iY+pWin->iY)*_Columns)*2;
        lpR = ( LPWORD )&_lpVIOSMEM[ iAdr ];
        for( iX = 0; iX < pWin->iW; iX++ ) *(lpRows++) = *(lpR++);
    }
}
return pMem;
}

/*****
/* win_Restore : restore window contents */
/**-----*/
/* input : pMem - transmit prior address via win_Save */
/*          Memory units, which contain window information */
/*          (position, dimension, cursor), as well as */
/*          content. */
/*          bFree - Should the memory unit be released after */
/*          restoration? (bFree != 0 ) */
/*****
VOID _FP win_Restore( PBYTE pMem, BYTE bFree )
{
    LPWORD lpRows;
    PWINDOW pWinData;

```

```

win_GetVIOSMEM();
if( pMem )
{
    static INT iX, iY;
    pWinData = ( PWINDOW )pMem;
    lpRows = ( PWORD ) &pWinData[ 1 ];
    for( iY = 0; iY < pWinData->iH; iY++ )
    {
        static WORD iAdr;
        static LPWORD lpR;
        iAdr = (pWinData->iX+(iY+pWinData->iY)*_Columns)*2;
        lpR = ( LPWORD ) &_lpVIOSMEM[ iAdr ];
        for( iX = 0; iX < pWinData->iW; iX++ )    *(lpR++) = *(lpRows++);
    }
    win_GotoXY( pWinData, pWinData->iCX, pWinData->iCY );

    if( bFree ) free( pMem );
}
}

/*****
/* win_GetActScreenSettings : Determine current screen
/*
/* settings
**-----**
/* input : pWin - address of the WINDOW structure, which should
/*
/* record screen settings
**-----**
/* Info : current video mode and current video page are not
/*
/* saved in WINDOW structure.
**
*****/
VOID _FP win_GetScreenSettings( PWINDOW pWin )
{ static union _REGS regs;

```



```

win_GetVIOSMEM();

win_Init( pWin, 0, 0, _Columns, 25,
          0x00, 0x00, WIN_HASCURSOR | WIN_ACTIVE );

regs.h.ah = VIOGETCURSOR;
regs.h.bh = VIOPAGE;
_int86( VIOINT, &regs, &regs );

pWin->iCX = regs.h.dl;
pWin->iCY = regs.h.dh;
}

/*****
/* win_ScrollUp : scroll window contents up          */
/**-----*/
/* input : pWin   window address                    */
/*          NumCols - number of lines to scroll up    */
/**-----*/
/* Info : The area made clear by scrolling is filled */
/*          with spaces in the current text attribute. */
/*****/
VOID _FP win_ScrollUp( PWINDOW pWin, INT NumCols )
{
    static INT iX, iY;
    static LPWORD lpRow, lpRowDest;
    static WINDOW Clear;

    win_GetVIOSMEM();

    for( iY = NumCols; iY < pWin->iH; iY++ )
    {
        static WORD iAdr, iAdr;
        iAdr = (pWin->iX+((iY-NumCols)+pWin->iY)*_Columns)*2;

```

```

        iAdr = (pWin->iX+(iY+pWin->iY)*_Columns)*2 ;
        lpRowDest = ( LPWORD ) &_lpVIOGMEM[ iAdr ];
        lpRow      = ( LPWORD ) &_lpVIOGMEM[ iAdr];
        for( iX = 0; iX < pWin->iW; iX++ ) *( lpRowDest++ ) = *( lpRow++ );
    }
    win_Init( &Clear, pWin->iX, pWin->iY + ( pWin->iH - NumCols ),
              pWin->iW, NumCols, pWin->iLoAttr, pWin->iHiAttr,
              WIN_CRLF | WIN_SCROLL );
    win_Clr( &Clear );
}

/*****
/* _win_PrINT : Text print out engine                                     */
/**-----**
/* input : pWin - Window address, in which text                         */
/*          should be printed out.                                       */
/*          pText - address for the text to be printed out              */
/*          iCnt - number of characters to be printed out               */
/*          (-1 = entire text)                                           */
*****/
VOID _FP _win_Print( PWINDOW pWin, PCHAR pText, INT iCnt )
{
    INT i;
    static LPBYTE lpRow;
    static WORD   iAdr;
    i = 0;

    win_GetVIOGMEM();

    iAdr = (pWin->iX+pWin->iCX+(pWin->iCY+pWin->iY)*_Columns)*2;
    lpRow = &_lpVIOGMEM[ iAdr ];

    while( ( pText[ i ] ) && iCnt )

```

```

{ static CHAR c;
  c = pText[ i ];
  switch( c )
  { default:                                     /* print out individual characters */
    *( lpRow++ ) = c;
    *( lpRow++ ) = pWin->iAttr;

    pWin->iCX++;
    if( pWin->iCX < pWin->iW ) break;          /* end of line reached ? */
  case '\r':                                   /* CR = CR + LF ? */
    if( !(pWin->uFlags & WIN_CRLF) && ( pText[ i ]== '\r' ) )
    {
      pWin->iCX = 0;                          /* at the beginning of the current line */
      iAdr = (pWin->iX+pWin->iCX+(pWin->iCY+pWin->iY)
        *_Columns )*2;
      lpRow = &_lpVIOSMEM[ iAdr ];
      break;
    }
  case '\n':                                   /* LF */
    while( pWin->iCX < pWin->iW )              /* erase to the end of the line */
    {
      *( lpRow++ ) = ' ';
      *( lpRow++ ) = pWin->iAttr;
      pWin->iCX++;
    }
    pWin->iCY++;                               /* next line */
    if( pWin->iCY >= pWin->iH )                /* reached last line? */
    {
      pWin->iCY = pWin->iH - 1;
      if( pWin->uFlags & WIN_SCROLL )         /* scroll content */
        win_ScrollUp( pWin, 1 );
    }
  }
}

```

```

        pWin->iCX = 0;
        iAdr = (pWin->iX+pWin->iCX+(pWin->iCY+pWin->iY)
                *_Columns )*2;
        lpRow = &_lpVIOGMEM[ iAdr ];
        break;
    }
    i++;
    if( iCnt > 0 ) iCnt--;    /* decrease counter, if positive */
}
win_GotoXY( pWin, pWin->iCX, pWin->iCY );
}

/*****
/* win_Print : print C-String in window */
/**-----*/
/* input : pWin - window, in which string should be printed out */
/*          pText - string to be printed. */
/**-----*/
/* Info : This function utilizes _win_Print for printing. */
/*****
VOID _FP win_Print( PWINDOW pWin, PCHAR pText )
{
    _win_Print( pWin, pText, -1 );
}

/*****
/* win_PrintAT : print text at any point in the window */
/**-----*/
/* input : pWin - window, in which string should be printed */
/*          iX, iY - print position */
/*          pText - string to be printed. */
/**-----*/

```

```

VOID _FP win_PrintAt( PWINDOW pWin, INT iX, INT iY, PCHAR pText )
{
    win_GotoXY( pWin, iX, iY );
    _win_Print( pWin, pText, -1 );
}

/*****
/* _wprintf : text print out like printf in window or on the screen */
/**-----*/
/* input : pWin      - window address, in which the print out      */
/*                  should occur (optional)                        */
/*          pFormat   - format string address                      */
/*          ...       - optional parameters                        */
/**-----*/
/* Info : This function utilizes vprintf, or vsprintf, to print out */
/*         the format string onto the screen, or into a buffer,     */
/*         which eventually will be printed in a window.           */
*****/
VOID _FP win_printf( PWINDOW pWin, PCHAR pFormat, ... )
{ static va_list argptr;
  va_start( argptr, pFormat );
  if( pWin )
  { static CHAR Buffer[ 1024 ];          /* max. 1024 characters */
    vsprintf( Buffer, pFormat, argptr );
    win_Print( pWin, Buffer );
  }
  else vprintf( pFormat, argptr );
  va_end( argptr );
}

/*****
/* win_Beep : produce warning sound */

```

```

/*****
VOID _FP win_Beep( VOID )
{ WORD uFrq = 400;
  LONG l;

      /* Beep! */
  outp( 0x43, 0xB6 );
  outp( 0x42, LOBYTE( 1193180L / uFrq ) );
  outp( 0x42, HIBYTE( 1193180L / uFrq ) );
  outp( 0x61, inp( 0x61 ) | 0x03 );
  for( l = 0; l < 10000L; l++ );
  outp( 0x61, inp( 0x61 ) & ~0x03 );
}

/*****
/* win_OBJECTInitINT - connect object with INT-type information */
/**-----*/
/* input: pObject      - type free object */
/*          pData       - type information (PINTDATA) */
/*          x, y, w, h   - position, dimension of the object */
/*          pText        - text to be printed */
/*          iMin, iMax   - integer marginal value */
/*          pValue       - address of the variables, which accept INT */
/*****
VOID _FP win_OBJECTInitINT( POBJECT pObject, PINTDATA pData,
                          INT x, INT y, INT w, INT h,
                          PCHAR pText, INT iMin, INT iMax, PINT pValue )
{
  pObject->X = x;
  pObject->Y = y;
  pObject->W = w; /* not utilized (yet)*/
  pObject->H = h;
  pObject->iType = OT_INT;

```

```

pObject->pData = ( PVOID ) pData;
pData->iMin = iMin;
pData->iMax = iMax;
pData->pText = pText;
pData->pValue = pValue;
}

/*****
/* win_OBJECTInitBOOL - connect object with INT-type information */
/**-----**/
/* input: pObject      - type free object */
/*          pData       - type information (PBOOLDATA) */
/*          x, y, w, h   - position, dimension of the object */
/*          pText        - text to be printed */
/*          iDisplay     - type of print ( YES/NO, TRUE/FALSE, ON/OFF ) */
/*          pValue       - address of the variables, that accept INT */
/*****/
VOID _FP win_OBJECTInitBOOL( POBJECT pObject, PBOOLDATA pData,
    INT x, INT y, INT w, INT h,
    PCHAR pText, INT iDisplay, PINT pValue )
{
    pObject->X = x;
    pObject->Y = y;
    pObject->W = w; /* not utilized (yet) */
    pObject->H = h;
    pObject->iType = OT_BOOL;
    pObject->pData = ( PVOID ) pData;
    pData->pText = pText;
    pData->iDisplay = iDisplay;
    pData->pValue = pValue;
}

```

```

/*****
/* win_OBJECTPrint - Display object */
/**-----**/
/* input: pWin      - print window */
/*          pObject  - object to be printed */
/**-----**/
/* Info: In this function, the object types are determined in order */
/*          to output the objects according to their respective style. */
/*****
VOID _FP win_OBJECTPrint( PWINDOW pWin, POBJECT pObject )
{ static CHAR pBuffer[ 256 ];
  switch( pObject->iType )
  {
    case OT_INT:
    { PINTDATA pIntData;
      /* dereference object */
      pIntData = ( PINTDATA ) pObject->pData;
      sprintf( pBuffer, "%s%d", pIntData->pText, *pIntData->pValue );
      win_PrintAt( pWin, pObject->X, pObject->Y, pBuffer );
    }
    break;
    case OT_BOOL:
    { PBOOLDATA pBoolData;
      /* Dereference object */
      pBoolData = ( PBOOLDATA ) pObject->pData;
      switch( pBoolData->iDisplay )
      {
        case DT_TRUEFALSE:
          sprintf( pBuffer, "%s%s",
            pBoolData->pText,
            *pBoolData->pValue ? " TRUE":"FALSE" );
          break;

```



```

        case DT_ONOFF:
            sprintf( pBuffer, "%s%s",
                pBoolData->pText,
                *pBoolData->pValue ? " ON":"OFF" );
            break;
        case DT_YESNO:
            sprintf( pBuffer, "%s%s",
                pBoolData->pText,
                *pBoolData->pValue ? " YES":"NO" );
            break;
    }
    win_PrintAt( pWin, pObject->X, pObject->Y, pBuffer );
}
}
}

/*****
/* win_OBJECTQueryValue - Determine object value */
/**-----**/
/* input: pObject      - object whose value is to be determined */
/* output:      value, whose type depends on the object type, and thus */
/*              may need to be changed. */
/*****
LONG _FP win_OBJECTQueryValue( POBJECT pObject )
{
    switch( pObject->iType )
    {
        case OT_INT:
        { PINTDATA pIntData;
            /* dereference object */
            pIntData = ( PINTDATA ) pObject->pData;
            return *pIntData->pValue;
        }
    }
}

```

```

    }

    case OT_BOOL:
    { PBOOLDATA pBoolData;
      /* dereference object*/
      pBoolData = ( PBOOLDATA ) pObj->pData;
      return *pBoolData->pValue;
    }
  }
  return 0L;
}

/*****
/* win_OBJECTPrintArray - print object list */
**-----**
/* input: pWin      - print window */
/*          pOArray - object list ( Array ) */
/*          iNum     - number of objects */
/*          iAct     - number of current object */
*****/
VOID _FP win_OBJECTPrintArray( PWINDOW pWin,
                              OBJECT pOArray[],
                              INT iNum,
                              INT iAct )
{ INT i;

  for( i = 0; i < iNum; i++ ) /* display everything */
  {
    if( i == iAct ) win_HiVideo( pWin );
    else           win_LoVideo( pWin );
    win_OBJECTPrint( pWin, &pOArray[ i ] );
  }
}

```

```

}

/*****
/* win_OBJECTProcessKey - Manage/process keyboard input */
/**-----**/
/* input: pWin      - print window */
/*      pObject - object that is to process the keyboard input */
/*      iKey      - keyboard code */
/*****
VOID _FP win_OBJECTProcessKey( PWINDOW pWin, POBJECT pObject, INT iKey )
{
    switch( pObject->iType )
    {
        case OT_INT:
        { PINTDATA pIntData;
          /* dereference object */
          pIntData = ( PINTDATA ) pObject->pData;
          switch( iKey )
          {
              case '+': (*pIntData->pValue)++; break;
              case '-': (*pIntData->pValue)--; break;
          }
          if( (*pIntData->pValue) < pIntData->iMin )
              *pIntData->pValue = pIntData->iMax;
          if( (*pIntData->pValue) > pIntData->iMax )
              *pIntData->pValue = pIntData->iMin;

          win_HiVideo( pWin );
          win_OBJECTPrint( pWin, pObject );
        }
        break;
        case OT_BOOL:

```

```

    { PBOOLDATA pBoolData;
      /* dereference object */
      pBoolData = ( PBOOLDATA ) pObj->pData;
      switch( iKey )
      {
        case '+': case '-':
          if(*pBoolData->pValue) *pBoolData->pValue = 0;
          else                    *pBoolData->pValue = 0xFFFF;
        }
      win_HiVideo( pWin );
      win_OBJECTPrint( pWin, pObj );
    }
    break;
  }
}

/*****
/* win_OBJECTWantsKey - Can the object use keyboard input?          */
/**-----**/
/* input: pObj - object, which is to process keyboard input        */
/*          iKey   - keyboard code                                   */
/* output:  == 0 - key code cannot be used                          */
/*          <> 0 - object key code usable                            */
/*****
INT _FP win_OBJECTWantsKey( POBJECT pObj, INT iKey )
{
  switch( pObj->iType )
  {
    case OT_INT:
    case OT_BOOL:
    {
      switch( iKey )

```

```

        {
            case '+':
            case '-': return TRUE;
        }
    }
    break;
}
return FALSE;
}

/*****
/* win_OBJECTProcessArray - process object list (Dialog) */
/**-----**/
/* input: pWin      - print window */
/*          pOArray - object list */
/*          iNum     - number of objects */
/*          pFunc    - message function */
/**-----**/
/* Info: Use TAB and <SHIFT>+TAB to shift the input FOCUS. */
/*          The user function pFunc is called,in order to give the user */
/*          the opportunity to react immediately to new */
/*          object states. */
/*****
VOID _FP win_OBJECTProcessArray( PWINDOW pWin,
                                OBJECT pOArray[],
                                INT iNum,
                                VOID ( *lpFunc ) ( INT iNum,
                                INT msg,
                                INT iParam,
                                LONG lParam ) )
{
    INT iAct;
    INT iLoop;

```

```

iAct = 0;
win_OBJECTPrintArray( pWin, pOArray, iNum, iAct );

iLoop = TRUE;
while( iLoop )
{
    WORD c;
    LONG oldVal;
    c = _getch();

    if( !c ) c = MAKEWORD( _getch(), 0 );
    if( win_OBJECTWantsKey( &pOArray[ iAct ], c ) )
    {
        if( lpFunc ) lpFunc( iAct, MSG_PRECHANGE, 0, 0L );
        oldVal = win_OBJECTQueryValue( &pOArray[ iAct ] );
        win_OBJECTProcessKey( pWin, &pOArray[ iAct ], c );
        if( lpFunc ) lpFunc( iAct, MSG_CHANGED, 0, oldVal );
    }
    else
    {
        win_LoVideo( pWin );
        win_OBJECTPrint( pWin, &pOArray[ iAct ] );
        if( lpFunc ) lpFunc( iAct, MSG_LOSTFOCUS, 0, 0L );
        switch( c )
        {
            case KBD_ESC: iLoop = FALSE; break;
            case KBD_UP: iAct--; if( iAct < 0 ) iAct = iNum - 1; break;
            case KBD_DN: iAct++; if( iAct >= iNum ) iAct = 0; break;
        }
        win_HiVideo( pWin );
        win_OBJECTPrint( pWin, &pOArray[ iAct ] );
        if( lpFunc ) lpFunc( iAct, MSG_GOTFOCUS, 0, 0L );
    }
}

```

```
    }  
    if( lpFunc ) lpFunc( iAct, MSG_KEY, c, 0L );  
  }  
#endif
```