

```

/*****
*
*                               X M S C . C
*
**-----**
* Task           : Demonstrates accessing extended memory and the
*                 high memory area using XMS functions, as
*                 implemented through the HIMEM.SYS device driver,
*                 for example.
*
*                 All device drivers must be loaded into
*                 conventional memory prior to running this test.
*
**-----**
* Author          : Michael Tischer
* Developed on    : 07/27/90
* Last update     : 04/07/95
*
**-----**
* (MICROSOFT C)
* Compilation     : CL /AS /Zp /c xmsc.c
*                 LINK xmsc xmsca;
* Call            : xmsc
*
**-----**
* (BORLAND TURBO C)
* Compilation     : Create a project file using the following:
*                 xmsc.c
*                 xmsca.obj
*****/

/*-- Link include files -----*/

#include <dos.h>                                /* for interrupt calls */

#ifdef __TURBOC__
#include <alloc.h>
#else
#include <malloc.h>
#endif

/*-- Constants -----*/

#define ERR_NOERR                0x00                /* No error */
#define ERR_NOTIMPLEMENTED      0x80                /* Specified function not known */
#define ERR_VDISKFOUND          0x81                /* VDISK-RAMDISK detected */
#define ERR_A20                 0x82                /* Error on A20 handler */
#define ERR_GENERAL              0x8E                /* General driver error */
#define ERR_UNRECOVERABLE        0x8F                /* Unrecoverable error */
#define ERR_HMANOTEXIST          0x90                /* HMA does not exist */
#define ERR_HMAINUSE             0x91                /* HMA already in use */
#define ERR_HMAMINSIZE           0x92                /* Not enough space in HMA */
#define ERR_HMANOTALLOCED        0x93                /* HMA not allocated */
#define ERR_A20STILLON           0x94                /* A20 handler still on */
#define ERR_OUTOMEMORY           0xA0                /* Out of extended memory */
#define ERR_OUTOHANDLES          0xA1                /* All XMS handles in use */
#define ERR_INVALIDHANDLE        0xA2                /* Invalid handle */
#define ERR_SHINVALID            0xA3                /* Source handle invalid */
#define ERR_SOINVALID            0xA4                /* Source offset invalid */
#define ERR_DHINVALID            0xA5                /* Destination handle invalid */
#define ERR_DOINVALID            0xA6                /* Destination offset invalid */
#define ERR_LENINVALID           0xA7                /* Invalid length for move function */
#define ERR_OVERLAP              0xA8                /* Illegal overlapping */
#define ERR_PARITY               0xA9                /* Parity error */
#define ERR_EMBUNLOCKED          0xAA                /* UMB is unlocked */
#define ERR_EMBLOCKED            0xAB                /* UMB is still locked */
#define ERR_LOCKOVERFLOW         0xAC                /* UMB lock overflow */
#define ERR_LOCKFAIL             0xAD                /* UMB cannot be locked */
#define ERR_UMBSIZETOOBIG        0xB0                /* Smaller UMB available */
#define ERR_NOUMBS               0xB1                /* No more UMB available */
#define ERR_INVALIDUMB           0xB2                /* UMB segment address is invalid */

#define TRUE ( 0 == 0 )
#define FALSE ( 0 == 1 )

/*-- Macros -----*/

#ifndef MK_FP
#define MK_FP(seg,ofs) \
((void far *) (((unsigned long)(seg) << 16) | (unsigned)(ofs)))

```

```

#endif

#define Hi(x) (((BYTE *) &x+1))          /* High byte of an int */
#define Lo(x) (((BYTE *) &x))           /* Low byte of an int */

/*-- Type declarations -----*/

typedef unsigned char BYTE;
typedef BYTE BOOL;
typedef unsigned WORD;

typedef struct                          /* Information for XMS call */
{
    WORD AX,                          /* Only registers AX, BX, DX and SI */
        BX,                          /* depending on the called function, */
        DX,                          /* along with a segment address */
        SI,
        Segment;
} XMSRegs;

typedef struct                          /* An extended memory move structure */
{
    long LenB;                        /* Number of bytes to be moved */
    int SHandle;                      /* Source handle */
    long SOffset;                     /* Source offset */
    int DHandle;                      /* Destination handle */
    long DOffset;                     /* Destination offset */
} EMMS;

/*-- External declarations -----*/

extern void XMSCall( BYTE FktNr, XMSRegs *Xr );

/*-- Global variables -----*/

void far * XMSPtr;    /* Pointer to the extended memory manager (XMM) */
BYTE XMSErr;         /* Error code of the last operation */

/*****
* XMSInit : Initializes the routines for calling the XMS functions
*-----*/
* Input   : None
* Output  : TRUE, if an XMS drive has been detected, otherwise FALSE
* Info    : - The call for this function must precede the calls for
*            all other procedures and functions from this program.
*****/

BOOL XMSInit( void )
{
    union REGS Regs;                  /* Processor registers for interrupt call */
    struct SREGS SRegs;               /* Segment registers */
    XMSRegs Xr;                      /* Registers for XMS call */

    Regs.x.ax = 0x4300;               /* Determine availability of XMS manager */
    int86( 0x2F, &Regs, &Regs );    /* Call DOS dispatcher */

    if ( Regs.h.al == 0x80 )           /* XMS manager found? */
    {                                  /* Yes */
        Regs.x.ax = 0x4310;           /* Determine entry point of XMM */
        int86x( 0x2F, &Regs, &Regs, &SRegs );
        XMSPtr = MK_FP( SRegs.es, Regs.x.bx ); /* Store addr.in global var */
        XMSErr = ERR_NOERR;           /* Still no error */
        return TRUE;                  /* Handler found, module initialized */
    }
    else                               /* No XMS-Handler installed */
        return FALSE;
}

/*****
* XMSQueryVer: Supplies the XMS version number and other status
*            information
*-----*/
* Input   : VerNo = Gets the version number after the functino call
*            (Format: 235 == 2.35)
*            RevNo = Gets the revision number after the function call
* Output  : TRUE, if an HMA is available, otherwise FALSE
*****/

```

```

*****/

BOOL XMSQueryVer( int * VerNr, int * RevNr)
{
    XMSRegs Xr;                                /* Registers for XMS call */

    XMSCall( 0, &Xr );                          /* Call XMS function #0 */
    *VerNr = Hi(Xr.AX)*100 + ( Lo(Xr.AX) >> 4 ) * 10 +
              ( Lo(Xr.AX) & 15 );
    *RevNr = Hi(Xr.BX)*100 + ( Lo(Xr.BX) >> 4 ) * 10 +
              ( Lo(Xr.BX) & 15 );
    return ( Xr.DX == 1 );
}

/*****
* XMSGetHMA : Gets the user access to the HMA.
*-----*
* Input      : LenB = Number of bytes to be allocated
* Info       : TSR programs should only request the memory size that is
*              truly required, while applications specify the value
*              0xFFFF.
* Output      : TRUE, if the HMA was made available, otherwise FALSE;
*****/

BOOL XMSGetHMA( WORD LenB )
{
    XMSRegs Xr;                                /* Registers for XMS call */

    Xr.DX = LenB;                              /* Pass length in register DX */
    XMSCall( 1, &Xr );                          /* Call XMS function #1 */
    return XMSErr == ERR_NOERR;
}

/*****
* XMSReleaseHMA : Releases the HMA, making it possible to pass it on
*                 to other programs.
*-----*
* None          : None
* Info          : - Call this procedure before ending a program when the
*                  HMA has been accessed beforehand by calling XMSGetHMA,
*                  since otherwise, the HMA cannot be used by programs
*                  called later.
*                  - This procedure causes the data stored in the HMA to be
*                  lost.
*****/

void XMSReleaseHMA( void )
{
    XMSRegs Xr;                                /* Registers for XMS call */

    XMSCall( 2, &Xr );                          /* Call XMS function #2 */
}

/*****
* XMSA20OnGlobal: Switches on A20 handler, making direct access
*                 to the HMA possible.
*-----*
* None          : None
* Info          : - On many computers, switching on the A20 handler is a
*                  relatively time-consuming process. Call this procedure
*                  only when necessary.
*****/

void XMSA20OnGlobal( void )
{
    XMSRegs Xr;                                /* Registers for XMS call */

    XMSCall( 3, &Xr );                          /* Call XMS function #3 */
}

/*****
* XMSA20OffGlobal: As a counterpart to the XMSA20OnGlobal procedure,
*                  this process switches A20 back off, so that direct
*                  access to the HMA is no longer possible.
*-----*
* Input          : None
*****/

```

```

* Info      : - Always call this procedure before ending a program,
*              in case A20 was switched on via a call for
*              XMSA20OnGlobal.
*****/
void XMSA20OffGlobal( void )
{
    XMSRegs Xr;                                /* Registers for XMS call */
    XMSCall( 4, &Xr );                         /* Call XMS function #4 */
}

/*****
* XMSA20OnLocal: See XMSA20OnGlobal
**-----**
* Input      : None
* Info      : - This local procedure differs from the global procedure
*              in that A20 is only switched on if hasn't been called
*              previously.
*****/
void XMSA20OnLocal( void )
{
    XMSRegs Xr;                                /* Registers for XMS call */
    XMSCall( 5, &Xr );                         /* Call XMS function #5 */
}

/*****
* XMSA20OffLocal : See XMSA29OffGlobal
**-----**
* Input      : None
* Info      : - This local procedure differs from the global procedure
*              in that A20 is only switched off if this hasn't already
*              taken place via a previous call.
*****/
void XMSA20OffLocal( void )
{
    XMSRegs Xr;                                /* Registers for XMS call */
    XMSCall( 6, &Xr );                         /* Call XMS function #6 */
}

/*****
* XMSIsA20On : Returns the status of the A20 handler
**-----**
* Input      : None
* Output     : TRUE, if A20 handler is switched on, otherwise FALSE
*****/
BOOL XMSIsA20On( void )
{
    XMSRegs Xr;                                /* Registers for XMS call */
    XMSCall( 7, &Xr );                         /* Call XMS function #7 */
    return ( Xr.AX == 1 );                     /* AX == 1 ---> Handler is free */
}

/*****
* XMSQueryFree : Returns the size of free extended memory and the
*               largest free block
**-----**
* Input      : TotFree: Gets the total size of free EM.
*              MaxBl  : Gets the size of the largest free block.
* Info      : - Both specifications in kilobytes
*              - The size of the HMA is not included in the count,
*              even if it hasn't yet been assigned to a program.
*****/
void XMSQueryFree( int * TotFree, int * MaxBl )
{
    XMSRegs Xr;                                /* Registers for XMS call */
    XMSCall( 8, &Xr );                         /* Call XMS function #8 */
    *TotFree = Xr.AX;                          /* Total size in AX */
}

```



[illegible]

```

Xr.DX = Handle;                                /* Handle of EMB */
Xr.BX = NewLenKB;                               /* New length in the BX register */
XMSCall( 15, &Xr );                             /* Call XMS function #15 */
return ( XMSErr == ERR_NOERR );
}

/*****
* XMSGetUMB : Allocates an upper memory block (UMB).
**-----**
* Input   : LenPara : Size of area to be allocated in paragraphs
*           of 16 bytes each
*           Seg      : Variable that gets the segment address of
*           the allocated UMB in successful cases
*           MaxPara  : Variable that specifies the length of the
*           largest available UMB in unsuccessful cases
* Output  : TRUE, if a UMB could be allocated, otherwise FALSE
* Info    : Warning! This function is not supported by all XMS
*           drivers and is extremely hardware-dependent.
*****/

BOOL XMSGetUMB( int LenPara, WORD * Seg, WORD * MaxPara )
{
    XMSRegs Xr;                                /* Registers for XMS call */

    Xr.DX = LenPara;                            /* Desired length to DX */
    XMSCall( 16, &Xr );                        /* Call XMS function #16 */
    *Seg = Xr.BX;                               /* Return segment address */
    *MaxPara = Xr.DX;                           /* Length of largest UMB */
    return ( XMSErr == ERR_NOERR );
}

/*****
* XMSFreeUMB : Releases UMB previously allocated by XMSGetUMB
**-----**
* Input   : Seg : Segment address of UMB being released
* Info    : Warning! This function is not supported by all XMS
*           drivers and is extremely hardware-dependent.
*****/

void XMSFreeUMB( WORD Seg )
{
    XMSRegs Xr;                                /* Registers for XMS call */

    Xr.DX = Seg;                               /* Segment address of UMB to DX */
    XMSCall( 17, &Xr );                       /* Call XMS function #17 */
}

/*-----*/
/*-- Test and Demo procedures --*/
/*-----*/

/*****
* HMA Test : Tests the availability of HMA and demonstrates its use
**-----**
* Input   : None
*****/

void HMA Test( void )
{
    BOOL A20;                                /* Current status of A20 handler */
    BYTE far * hmap;                          /* Pointer to HMA */
    WORD i,                                   /* Loop counter */
    err;                                       /* Number of errors in HMA access */

    printf( "HMA-Test - Please press a key to start the test..." );
    getch();
    printf( "\n\n" );

    /*-- Allocate HMA and test each memory location -----*/

    if ( XMSGetHMA(0xFFFF) )                 /* HMA acquired? */
    {                                           /* Yes */
        if ( ( A20 = XMSIsA20On() ) == FALSE ) /* Determine handler status */
            XMSA20OnGlobal();                 /* Switch it on now */
    }
}

```

```

hmap = MK_FP( 0xFFFF, 0x0010 ); /* Pointer to HMA */
err = 0; /* No errors up until now */
for ( i = 1; i < 65520; ++i, ++hmap )
{
    /* Test each single memory location */
    printf( "\r Testing Memory Location: %u", i );
    *hmap = i % 256; /* Write memory location */
    if ( *hmap != i % 256 ) /* And read out again */
    {
        /* Error! */
        printf( " ERROR!\n" );
        ++err;
    }
}

XMSReleaseHMA(); /* Release HMA */
if ( A20 == FALSE ) /* Was A20 handler on? */
    XMSA20OffGlobal(); /* No, switch it off */

printf( "\n" );
if ( err == 0 ) /* Evaluate results of test */
    printf( "HMA ok, no defective memory location.\n" );
else
    printf( "ATTENTION! %d defective memory locations detected " \
        "in HMA!\n", err );
}
else
    printf( "ATTENTION! No access to HMA possible.\n" );
}

/*****
* EMBTest : Tests extended memory and demonstrates the calls of
*           different XMS functions
**-----**
* Input   : None
*****/

void EMBTest( void )
{
    long Adr;
    BYTE * barp; /* Pointer to 1K buffer */
    int i, j, /* Loop counter */
        err, /* Number of errors in HMA access */
        Handle, /* Handle for access to EMB */
        TotFree, /* Size of total free extended memory */
        MaxBl; /* Largest free block */

    printf( "EMB Test - Please press a key to start the test..." );
    getch();
    printf( "\n" );

    XMSQueryFree( &TotFree, &MaxBl ); /* Determine size of extended mem */
    printf( "Total size of free extended memory (incl. HMA): %d KB\n",
        TotFree );
    printf( " Largest free block: %d KB\n",
        MaxBl );

    TotFree -= 64; /* Calculate actual size without HMA */
    if ( MaxBl >= TotFree ) /* Can the value be right? */
        MaxBl -= 64; /* No */

    if ( MaxBl > 0 ) /* Still enough memory free? */
    {
        /* Yes */
        Handle = XMSGetMem( MaxBl );
        printf( "%d KB allocated.\n", MaxBl );
        printf( "Handle = %d\n", Handle );
        Adr = XMSLock( Handle ); /* Determine address */
        XMSUnlock( Handle ); /* Unlock again */
        printf( "Start address = %ld (%d KB)\n", Adr, Adr >> 10 );

        barp = malloc( 1024 ); /* Buffer to heap allocated */
        err = 0; /* No errors up to now */

        /*-- Execute allocated EMB KB for KB and test -----*/

        for ( i = 0; i < MaxBl; ++i )

```



```

{
    printf( "\rKB Test: %d", i+1 );
    memset( barp, i % 255, 1024 );
    XMSCopy( 0, (long) ((void far *) barp),
             Handle, (long) i*1024, 512 );
    memset( barp, 255, 1024 );
    XMSCopy( Handle, (long) i*1024, 0,
             (long) ((void far *) barp), 512 );

    /*-- Compare copied buffer with expected result -----*/

    for ( j = 0; j < 1024; ++j )
        if ( *(barp+j) != i % 255 )
            {
                printf( " ERROR!\n" );
                ++err;
                break;
            }
}

printf( "\n" );
if ( err == 0 )
    printf( "EMB ok, none of the tested 1K blocks were " \
            "defective.\n");
else
    printf( "ATTENTION! %d defective 1K blocks detected in EMB\n",
            err );

free( barp );
XMSFreeMem( Handle );
}

/* Release buffer again */
/* Release EMB again */

}

/*****
*                               M A I N   P R O G R A M
*                               *****/

void main( void )
{
    int VerNr,
        RevNr,
        i;

    /* Version number */
    /* Revision number */
    /* Loop counter */

    for ( i = 1; i < 25; ++i )
        printf ( "\n" );

    printf("XMSC - XMS Demo program by Michael Tischer\n\n" );
    if ( XMSInit() )
    {
        if ( XMSQueryVer( &VerNr, &RevNr ) )
            printf( "Access to HMA possible.\n" );
        else
            printf( "No access to HMA.\n" );
        printf( "XMS version number: %d.%d\n", VerNr / 100, VerNr % 100 );
        printf( "Revision number   : %d.%d\n\n", RevNr / 100, RevNr % 100 );
        HMATest();
        printf( "\n" );
        EMBTest();
    }
    else
        printf( "No XMS driver installed!\n" );
}

```