

```

/*****
/*          C D U T I L . C          */
**/
/*-----**/
/* Task          : Utilities for MSCDEX programming */
/*               : Helper, Wrapper, Interfaces      */
/*-----**/
/* Author         : Michael Tischer / Bruno Jennrich */
/* Developed on    : 04/08/94                        */
/* Last update     : 04/18/94                        */
/*-----**/
/* COMPILER OPT.  : 1 Byte Struct Member Alignment! */
/* COMPILER       : Borland C++ 3.1, Microsoft Visual C++ 1.5 */
/*****
#ifdef __CDUTIL_C          /* CDUTIL.C can also #be included */
#define __CDUTIL_C

#include <dos.h>
#include <process.h>
#include <stdlib.h>

#include "types.h"
#include "cddev.h"
#include "cdutil.h"

/*****
/* Time2Frame : Converts time to number of frames (75 Frames = 1 sec) */
**/
/*-----**/
/* Input      : iMin   - Minute ( 0 - 59+) */
/*             iSec    - Second ( 0 - 59 ) */
/*             iFrame  - Frame   ( 0 - 75 ) */
/* Output     : Number of frames */
/*****
LONG Time2Frame( INT iMin, INT iSec, INT iFrame )
{
    return iMin * 60L * 75L + iSec * 75L + iFrame;
}

/*****
/* Time2HSG : Converts time to HSG-Address */
**/
/*-----**/
/* Input      : iMin   - Minute ( 0 - 59+) */
/*             iSec    - Second ( 0 - 59 ) */
/*             iFrame  - Frame   ( 0 - 75 ) */
/* Output     : Converted HSG address */
/*****
LONG Time2HSG( INT iMin, INT iSec, INT iFrame )
{
    return Time2Frame( iMin, iSec, iFrame ) - 150L;
}

/*****
/* Time2REDBOOK : Converts time to REDBOOK address */
**/
/*-----**/
/* Input      : iMin   - Minute ( 0 - 59+) */
/*             iFrame  - Frame   ( 0 - 75 ) */
/* Output     : Converted REDBOOK address */
/*****
LONG Time2REDBOOK( INT iMin, INT iSec, INT iFrame )
{
    return ( ( ( LONG ) iMin   << 16 ) |
              ( ( LONG ) iSec   << 8  ) |
              ( ( LONG ) iFrame << 0 ) );
}

/*****
/* FrameTime : Splits number of frames (sectors) into minute, */
/*            : second, frame. */
**/
/*-----**/
/* Input      : lFrames - number of frames (sectors) to be converted */
/*             lpiMin   - Address of an INT variable for receiving the */
/*                     : minutes */
/*             lpiSec   - ditto for seconds */
/*             lpiFrame - ditto for frames */
/*****
VOID Frame2Time( LONG lFrame, LPINT lpiMin,
                LPINT lpiSec,

```

```

                                LPINT lpiFrame )
{
    *lpiMin = ( INT ) ( lFrame / ( 60L * 75L ) );
    lFrame -= *lpiMin * 60L * 75L;
    *lpiSec = ( INT ) ( lFrame / 75L );
    lFrame -= *lpiSec * 75L;
    *lpiFrame = ( INT ) lFrame;
}

/*****
/* REDBOOK2Time : Splits REDBOOK address into minute, second, frame */
**-----**/
/* Input : lREDBOOK - REDBOOK address to be converted */
/*          lpiMin   - Address of an INT variable for receiving the */
/*                  minutes */
/*          lpiSec   - ditto for seconds */
/*          lpiFrame - ditto for frames */
/*****
VOID REDBOOK2Time( LONG lREDBOOK, LPINT lpiMin,
                  LPINT lpiSec,
                  LPINT lpiFrame )
{
    *lpiFrame = ( INT ) ( lREDBOOK >> 0 ) & 0x00FF;
    *lpiSec   = ( INT ) ( lREDBOOK >> 8 ) & 0x00FF;
    *lpiMin   = ( INT ) ( lREDBOOK >> 16 ) & 0x00FF;
}

/*****
/* HSG2Time : Splits HSG address into minute, second, frame. */
**-----**/
/* Input : lHSG      - HSG address to be converted */
/*          lpiMin    - Address of an INT variable for receiving the */
/*                  minutes */
/*          lpiSec    - ditto for seconds */
/*          lpiFrame  - ditto for frames */
/*****
VOID HSG2Time( LONG lHSG, LPINT lpiMin,
              LPINT lpiSec,
              LPINT lpiFrame )
{
    Frame2Time( lHSG + 150, lpiMin, lpiSec, lpiFrame );
}

/*****
/* REDBOOK2HSG : Convert REDBOOK address to HSG address */
**-----**/
/* Input : lREDBOOK - REDBOOK address to be converted */
/* Output : Converted HSG address */
/*****
LONG REDBOOK2HSG( LONG lREDBOOK )
{
    INT iMin, iSec, iFrame;
    REDBOOK2Time( lREDBOOK, &iMin, &iSec, &iFrame );
    return Time2HSG( iMin, iSec, iFrame );
}

/*****
/* HSG2REDBOOK : Convert HSG address to REDBOOK address */
**-----**/
/* Input : lHSG - HSG address to be converted */
/* Output : Converted REDBOOK address */
/*****
LONG HSG2REDBOOK( LONG lHSG )
{
    INT iMin, iSec, iFrame;
    HSG2Time( lHSG, &iMin, &iSec, &iFrame );
    return Time2REDBOOK( iMin, iSec, iFrame );
}

/*- MSCDEX Interface Functions -----*/

/*****
/* MSCDEX_GetNumberOfDriveLetters : Get number of supported */
/*                                CD-ROM drives. */
**-----**/
/* Input : lpiNumber      - Address of an INT that is to receive the */
/*                                number of drives (Output parameter) */
/*          lpiStartLetter - ASCII code of first drive letter */

```

```

/*          to be used by a CD-ROM drive.          */
/*          */
/* Output : TRUE - Operation successful              */
/*          FALSE - Operation failed ( Error in _doserrno ) */
/*-----*/
/* Info : This function can be used to find out      */
/*          whether CD-ROM drives are supported or whether */
/*          MSCDEX has been installed. Prior to the call, BX is set */
/*          to 0. If this register is still null after the call */
/*          it means no CD-ROM drive is supported. */
/*-----*/
INT MSCDEX_GetNumberOfDriveLetters( LPINT lpiNumber,
                                   LPINT lpiStartLetter )
{ union _REGS regs; /* AX = 0x1500, BX = 0 */
  regs.x.ax = MSCDEX_MUXCODE( MSCDEX_GET_NO_OF_DRIVE_LETTERS );
  regs.x.bx = 0;
  _int86( DOS_MULTIPLEX_INT, &regs, &regs );
  if( !regs.x.cflag ) /* Carry flag set ? */
  {
    *lpiNumber = regs.x.bx;
    *lpiStartLetter = regs.x.cx;
    return TRUE;
  }
  *lpiNumber = 0;
  *lpiStartLetter = 0;
  return FALSE;
}

/*-----*/
/* MSCDEX_Installed : MSCDEX installed ?          */
/*-----*/
/* Info : This function tests whether the number of supported */
/*          drives differs from 0. */
/*-----*/
INT MSCDEX_Installed( VOID )
{ INT iNumber, iStartLetter;
  iNumber = 0;
  MSCDEX_GetNumberOfDriveLetters( &iNumber,
                                  &iStartLetter );
  return ( iNumber == 0 ) ? FALSE : TRUE;
}

/*-----*/
/* MSCDEX_GetCDRomDriveDeviceList : Get information about */
/*          connected CD-ROM drives. */
/*-----*/
/* Input : lpDevElement - Address of a DevElement array, that */
/*          is to receive the information of ALL */
/*          connected drives. */
/* Output : TRUE - Operation successful */
/*          FALSE - Operation failed ( Error in _doserrno ) */
/*-----*/
/* Info : After the call for this function a DevElement contains the */
/*          address of the device header for the specified SubUnit. */
/*          The array passed to this function must have room for all */
/*          connected (or supported) CD-ROM drives. */
/*          The safest method consists of leaving space for 26 */
/*          elements (drive letters A: to Z:) */
/*          and accessing the first elements. */
/*-----*/
INT MSCDEX_GetCDRomDriveDeviceList( LPDevElement lpDevElement )
{ union _REGS regs; /* AX = 0x1501 */
  struct _SREGS sregs; /* ES:BX = Address of the buffer */
  regs.x.ax = MSCDEX_MUXCODE( MSCDEX_GET_DRIVE_DEVICE_LIST );
  sregs.es = FP_SEG( lpDevElement );
  regs.x.bx = FP_OFF( lpDevElement );
  _int86( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
  return ( regs.x.cflag == 0 ) ? TRUE : FALSE;
}

/*-----*/
/* MSCDEX_GetCopyrightFilename : Get name of copyright file */
/*          of a CD. */
/*-----*/
/* Input : iCD_Drive_Letter - Drive ID (0=A, 1=B, 2=C etc.) */
/*          lpName - Address of a 38 character */

```

```

/*          Buffer for the name.          */
/* Output : TRUE  - Operation successful          */
/*          FALSE - Operation failed ( Error in _doserrno )          */
/*-----*/
/* Info: Although 38 bytes are available for the filename          */
/*        only the first 11 characters (8.3) are used.          */
/*-----*/
INT MSCDEX_GetCopyrightFilename( INT      iCD_Drive_Letter,
                                LPCHAR    lpName )
{ union _REGS regs;              /* AX = 0x1502, CX = Drive letter */
  struct _SREGS sregs;           /* ES:BX = Address of name */
  regs.x.ax = MSCDEX_MUXCODE( MSCDEX_GET_COPYRIGHT_FILENAME );
  sregs.es = FP_SEG( lpName );
  regs.x.bx = FP_OFF( lpName );
  regs.x.cx = iCD_Drive_Letter;
  _int86x( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
  return ( regs.x.cflag == 0 ) ? TRUE : FALSE;
}

/*-----*/
/* MSCDEX_GetAbstractFilename : Get name of abstract file          */
/*          of a CD.          */
/*-----*/
/* Input  : iCD_Drive_Letter - Drive ID      (0=A, 1=B, 2=C etc.) */
/*          lpName           - Address of a 38 character          */
/*          buffer for the name.          */
/* Output : TRUE  - Operation successful          */
/*          FALSE - Operation failed ( Error in _doserrno )          */
/*-----*/
/* Info: Although 38 bytes are available for the filename          */
/*        only the first 11 characters (8.3) are used.          */
/*-----*/
INT MSCDEX_GetAbstractFilename( INT      iCD_Drive_Letter,
                                LPCHAR    lpName )
{ union _REGS regs;              /* AX = 0x1503, CX = Drive letter */
  struct _SREGS sregs;           /* ES:BX = Address of name */
  regs.x.ax = MSCDEX_MUXCODE( MSCDEX_GET_ABSTRACT_FILENAME );
  sregs.es = FP_SEG( lpName );
  regs.x.bx = FP_OFF( lpName );
  regs.x.cx = iCD_Drive_Letter;
  _int86x( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
  return ( regs.x.cflag == 0 ) ? TRUE : FALSE;
}

/*-----*/
/* MSCDEX_GetBibDocFilename : Get name of bibliographic          */
/*          documentation file of a CD.          */
/*-----*/
/* Input  : iCD_Drive_Letter - Drive ID (0=A, 1=B, 2=C etc.)      */
/*          lpName           - Address of a 38 character          */
/*          buffer for the name.          */
/* Output : TRUE  - Operation successful          */
/*          FALSE - Operation failed ( Error in _doserrno )          */
/*-----*/
/* Info: Although 38 bytes are available for the filename          */
/*        only the first 11 characters (8.3) are used.          */
/*-----*/
INT MSCDEX_GetBibDocFilename( INT      iCD_Drive_Letter,
                              LPCHAR    lpName )
{ union _REGS regs;              /* AX = 0x1504, CX = Drive letter */
  struct _SREGS sregs;           /* ES:BX = Address of name */
  regs.x.ax = MSCDEX_MUXCODE( MSCDEX_GET_BIB_DOC_FILENAME );
  sregs.es = FP_SEG( lpName );
  regs.x.bx = FP_OFF( lpName );
  regs.x.cx = iCD_Drive_Letter;
  _int86x( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
  return ( regs.x.cflag == 0 ) ? TRUE : FALSE;
}

/*-----*/
/* MSCDEX_ReadVTOC : Read Volume Table of Contents          */
/*-----*/
/* Input  : iCD_Drive_Letter - Drive ID (0=A, 1=B, 2=C etc.)      */
/*          lpSector         - Address of a 2048 character          */
/*          buffer for a sector.          */
/*          iNumSec          - Number of TOC sector to be read.      */

```

```

/* Output : <> 0 - Operation successful */
/*          0x0101 - Read sector = standard VTOC sector */
/*          0x01FF - Read sector = last VTOC sector */
/*          0x0100 - Read sector = other VTOC sector */
/*          0 - Error (s._doserrno) */
/*****/
INT MSCDEX_ReadVTOC( INT iCD_Drive_Letter,
                    LPBYTE lpSector,
                    INT iNumSec )
{ union _REGS regs; /* AX = 0x1505, CX = Drive letter */
  struct _SREGS sregs; /* ES:BX = Address of the name */
  regs.x.ax = MSCDEX_MUXCODE( MSCDEX_READ_VTOC );
  sregs.es = FP_SEG( lpSector );
  regs.x.bx = FP_OFF( lpSector );
  regs.x.cx = iCD_Drive_Letter;
  regs.x.dx = iNumSec;
  _int86( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
  return ( regs.x.cflag == 0 ) ? ( 0x0100 | regs.h.al ) : 0;
}

/*****/
/* MSCDEX_DebugOn : Enable debugging. */
/* (only for MSCDEX debug version) */
/*****/
/* Output : TRUE - Operation successful */
/*          FALSE - Operation failed ( Error in _doserrno ) */
/*****/
INT MSCDEX_DebugOn( VOID )
{ union _REGS regs; /* AX = 0x1506 */
  regs.x.ax = MSCDEX_MUXCODE( MSCDEX_DEBUG_ON );
  _int86( DOS_MULTIPLEX_INT, &regs, &regs );
  return ( regs.x.cflag == 0 ) ? TRUE : FALSE;
}

/*****/
/* MSCDEX_DebugOff : Disable debugging. */
/* (only for MSCDEX debug version) */
/*****/
/* Output : TRUE - Operation successful */
/*          FALSE - Operation failed ( Error in _doserrno ) */
/*****/
INT MSCDEX_DebugOff( VOID )
{ union _REGS regs; /* AX = 0x1507 */
  regs.x.ax = MSCDEX_MUXCODE( MSCDEX_DEBUG_OFF );
  _int86( DOS_MULTIPLEX_INT, &regs, &regs );
  return ( regs.x.cflag == 0 );
}

/*****/
/* MSCDEX_AbsoluteRead : Read data from sector(s) */
/*****/
/* Input : iCD_Drive_Letter - Drive ID (0=A, 1=B, 2=C etc.) */
/*          lpSector - Address of 2048 character */
/*          buffer for sectors. */
/*          iSecCnt - Number of sectors to be read. */
/*          lSecStart - Number of first sector to be read */
/* Output : TRUE - Operation successful */
/*          FALSE - Operation failed ( Error in _doserrno ) */
/*****/
/* Info: A 2048 character buffer must be provided for each */
/*        sector to be read. */
/*****/
INT MSCDEX_AbsoluteRead( INT iCD_Drive_Letter,
                        LPBYTE lpSector,
                        INT iSecCnt,
                        LONG lSecStart )
{ union _REGS regs; /* AX = 0x1508, CX = Drive letter */
  struct _SREGS sregs; /* DX = Number of sectors to be read */
                      /* ES:BX = Address of the read buffer */
                      /* SI:DI = Address of the first sector */

  regs.x.ax = MSCDEX_MUXCODE( MSCDEX_ABSOLUTE_DISK_READ );
  sregs.es = FP_SEG( lpSector );
  regs.x.bx = FP_OFF( lpSector );
  regs.x.cx = iCD_Drive_Letter;
  regs.x.dx = iSecCnt;

```

```

regs.x.si = HIWORD( lSecStart );
regs.x.di = LOWORD( lSecStart );
_int86x( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
return ( regs.x.cflag == 0 ) ? TRUE : FALSE;
}

/*****
/* MSCDEX_AbsoluteWrite: Write data to sector(s) */
/*-----**/
/* Input : iCD_Drive_Letter - Drive ID (0=A, 1=B, 2=C etc.) */
/*         lpSector          - Address of 2048 character buffer */
/*                               that contains the data. */
/*         iSecCnt           - Number of sectors to be written. */
/*         lSecStart         - Number of first sector to be written */
/* Output : TRUE - Operation successful */
/*         FALSE - Operation failed ( Error in _doserrno ) */
/*-----**/
/* Info: A 2048 character buffer must be provided */
/*         for each sector. */
*****/
INT MSCDEX_AbsoluteWrite( INT iCD_Drive_Letter,
                          LPBYTE lpSector,
                          INT iSecCnt,
                          LONG lSecStart )
{ union _REGS regs; /* AX = 0x1509, CX = Drive letter */
  struct _SREGS sregs; /* DX = Number of sectors to be written */
                          /* ES:BX = Address of write buffer */
                          /* SI:DI = Address of first sector */

  regs.x.ax = MSCDEX_MUXCODE( MSCDEX_ABSOLUTE_DISK_WRITE );
  sregs.es = FP_SEG( lpSector );
  regs.x.bx = FP_OFF( lpSector );
  regs.x.cx = iCD_Drive_Letter;
  regs.x.dx = iSecCnt;
  regs.x.si = HIWORD( lSecStart );
  regs.x.di = LOWORD( lSecStart );
  _int86x( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
  return ( regs.x.cflag == 0 ) ? TRUE : FALSE;
}

/*****
/* MSCDEX200_CDROMDriveCheck : Is the drive a CD-ROM drive? */
/*-----**/
/* Input : iCD_Drive_Letter - Drive ID (0=A, 1=B, 2=C etc.) */
/* Output : TRUE - Operation successful */
/*         FALSE - Operation failed ( Error in _doserrno ) */
*****/
INT MSCDEX200_CDROMDriveCheck( INT iCD_Drive_Letter )
{ union _REGS regs; /* AX = 0x150B, CX = Drive letter */

  regs.x.ax = MSCDEX_MUXCODE( MSCDEX200_CD_ROM_DRIVE_CHECK );
  regs.x.cx = iCD_Drive_Letter;
  regs.x.bx = 0;
  _int86( DOS_MULTIPLEX_INT, &regs, &regs );
  return ( ( regs.x.bx == 0xADAD ) && ( regs.x.ax != 0 ) ) ?
    TRUE : FALSE;
}

/*****
/* MSCDEX200_GetVersion : Get MSCDEX version */
/*-----**/
/* Output : Version number (Major.Minor) */
/* Info : This function has only been in existence since MSCDEX */
/*         V2.00! Smaller version numbers are specified as 1.00! */
*****/
INT MSCDEX200_GetVersion( VOID )
{ union _REGS regs; /* AX = 0x150C */
  regs.x.ax = MSCDEX_MUXCODE( MSCDEX200_GET_VERSION );
  regs.x.bx = 0; /* Clear first BX */
  _int86( DOS_MULTIPLEX_INT, &regs, &regs );
  if( ( regs.x.bx ) && ( regs.x.cflag == 0 ) ) return regs.x.bx;
  return ( regs.x.cflag == 0 ) ? 0x0100 : -1; /* V1.00 or Error */
}

/*****

```

```

/* MSCDEX200_GetCDromDriveLetters : Get drive letters          */
/*                                used by ALL CD-ROM drives.    */
/*-----**/
/* Input   : lpLetters - Address of character array, that gets */
/*                                drive letters.                */
/*                                (s. GetNumberOfDriveLetters )  */
/* Output  : TRUE  - Operation successful                       */
/*          FALSE - Operation failed ( Error in _doserrno )    */
/*-----**/
*****
INT MSCDEX200_GetCDromDriveLetters( LPCHAR lpLetters )
{ union _REGS regs;                                /* AX = 0x150D */
  struct _SREGS sregs;                             /* ES:BX = Address of array */

  regs.x.ax = MSCDEX_MUXCODE( MSCDEX200_GET_DRIVE_LETTERS );
  sregs.es  = FP_SEG( lpLetters );
  regs.x.bx = FP_OFF( lpLetters );
  _int86x( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
  return ( regs.x.cflag == 0 ) ? TRUE : FALSE;
}

/*-----**/
/* MSCDEX200_GetVDPPreference : Get Volume Descriptor Preference */
/*-----**/
/* Input   : iCD_Drive_Letter - Drive ID (0=A, 1=B, 2=C etc.) */
/*          lpiPreference      - Address of the INT, in which   */
/*                                Preference is to be stored.     */
/* Output  : TRUE  - Operation successful                       */
/*          FALSE - Operation failed ( Error in _doserrno )    */
/*-----**/
*****
INT MSCDEX200_GetVDPPreference( INT iCD_Drive_Letter,
                               LPINT lpiPreference )
{ union _REGS regs;                                /* AX = 0x150D */
  struct _SREGS sregs;                             /* ES:BX = Address of array */

  regs.x.ax = MSCDEX_MUXCODE( MSCDEX200_GETSET_VD_PREFERENCE );
  regs.x.cx = iCD_Drive_Letter;
  regs.x.bx = 0x0000;                               /* 0 = Get Preference */
  regs.x.dx = 0;                                     /* Clear result register as precaution */
  _int86x( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
  *lpiPreference = regs.x.dx;
  return ( regs.x.cflag == 0 ) ? TRUE : FALSE;
}

/*-----**/
/* MSCDEX200_SetVDPPreference : Set Volume Descriptor Preference */
/*-----**/
/* Input   : iCD_Drive_Letter - Drive ID (0=A, 1=B, 2=C etc.) */
/*          iPreference        - New Preference                 */
/* Output  : TRUE  - Operation successful                       */
/*          FALSE - Operation failed ( Error in _doserrno )    */
/*-----**/
*****
INT MSCDEX200_SetVDPPreference( INT iCD_Drive_Letter,
                               INT iPreference )
{ union _REGS regs;                                /* AX = 0x150D */
  struct _SREGS sregs;                             /* ES:BX = Address of array */

  regs.x.ax = MSCDEX_MUXCODE( MSCDEX200_GETSET_VD_PREFERENCE );
  regs.x.cx = iCD_Drive_Letter;
  regs.x.bx = 0x0001;                               /* 1 = Set Preference */
  regs.x.dx = iPreference;
  _int86x( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
  return ( regs.x.cflag == 0 ) ? TRUE : FALSE;
}

/*-----**/
/* MSCDEX200_GetDirectoryEntry : Get file information          */
/*-----**/
/* Input   : iCD_Drive_Letter - Drive ID (0=A, 1=B, 2=C etc.) */
/*          iCopyFlag          - 0 - direct copy, 1 - copy to struct */
/*          lpPathName         - Path name of file                */
/*          lpData             - Address to which data are to be copied */
/*          lpiHSG             - Directory data HSG(<>0) or ISO(=0) */
/* Output  : TRUE  - Operation successful                       */
/*          FALSE - Operation failed ( Error in _doserrno )    */
/*-----**/

```

```

/* Info : Allocated buffer should be at least 255 bytes */
/* (copy to struct) or 280 bytes (direct copy) large. */
/* With direct copy the programmer is responsible for decoding */
/* the structure data. The Int variable indicates which of the */
/* two types of structure (HSG_ENTRY or ISO_ENTRY) you are */
/* dealing with The lpiHSG references the Int variable. */
/*****
INT MSCDEX200_GetDirectoryEntry( INT iCD_Drive_Letter,
                                INT iCopyFlag,
                                LPCHAR lpPathName,
                                LPVOID lpData,
                                LPINT lpiHSG )
{ union _REGS regs; /* AX = 0x150F, CL = CD_Drive, CH = copy flags */
  struct _SREGS sregs; /* ES:BX = Path name, SI:DI = Data address */

  regs.x.ax = MSCDEX_MUXCODE( MSCDEX200_GET_DIRECTORY_ENTRY );
  regs.h.cl = ( BYTE )iCD_Drive_Letter;

  regs.h.ch = ( BYTE ) ( iCopyFlag ? 0x01 : 0x00 );
  sregs.es = FP_SEG( lpPathName );
  regs.x.bx = FP_OFF( lpPathName );
  regs.x.si = FP_SEG( lpData );
  regs.x.di = FP_OFF( lpData );
  _int86x( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
  *lpiHSG = regs.x.ax;
  return ( regs.x.cflag == 0 ) ? TRUE : FALSE;
}

/*****
/* MSCDEX_ReadWriteReq : Execute MSCDEX-IOCTLI_READ/WRITE Request */
/*-----*/
/* Input : iCD_Drive_Letter - ID of drive, to whose */
/* driver a request is to be sent. */
/* iCommand - Number of command to be executed */
/* lpControlBlock - Address of control block that */
/* defines READ/WRITE request. */
/* iControlBlockSize - Size of control block */
/* Output : Device-Status */
/*-----*/
/* Info : MSCDEX automatically fills up the SubUnit field when a */
/* driver is called that supports more than one drive. */
/* If MSCDEX does not start a driver command the SubUnit */
/* field must be set by the programmer (s. _CallStrategy() )*/
/*****
INT MSCDEX_ReadWriteReq( INT iCD_Drive_Letter,
                        INT iCommand,
                        LPVOID lpControlBlock,
                        INT iControlBlockSize )
{ MSCDEX_IOCTLIO IOCTLIO;
  IOCTLIO.RegHdr.bLength = sizeof( RequestHeader );
  IOCTLIO.RegHdr.bSubUnit = 0;
  IOCTLIO.RegHdr.bCommand = ( BYTE )iCommand;

  IOCTLIO.bMediaDescriptor = 0;
  IOCTLIO.lpTransferAddress = lpControlBlock;
  IOCTLIO.wTransferCount = iControlBlockSize;
  IOCTLIO.wStartingSectorNo = 0;
  IOCTLIO.lpVolumeID = NULL;

  return MSCDEX_SendDeviceRequest( iCD_Drive_Letter,
                                  &IOCTLIO.RegHdr );
}

/*****
/* MSCDEX210_SendDeviceRequest: Send device request to device */
/* driver */
/*-----*/
/* Input : iCD_Drive_Letter - Number of drive, to whose device */
/* driver a device request is to be sent. */
/* (0=A, 1=B, etc.) */
/* lpReqHdr - Address of device request header */
/* Output : Status of device after operation */
/* (s. RequestHeader.wStatus) */
/*-----*/
/* Info: MSCDEX uses this call to call the strategy routine of the */
/* device driver. */

```

```

/*****
INT MSCDEX210_SendDeviceRequest( INT          iCD_Drive_Letter,
                                LPRequestHeader lpReqHdr )
{ union _REGS regs;              /* AX = 0x1510, CX = Drive letter */
  struct _SREGS sregs;          /* ES:BX = Address of request header */

  regs.x.ax = MSCDEX_MUXCODE( MSCDEX210_SEND_DEVICE_REQUEST );
  regs.x.cx = iCD_Drive_Letter;
  sregs.es = FP_SEG( lpReqHdr );
  regs.x.bx = FP_OFF( lpReqHdr );
  _int86x( DOS_MULTIPLEX_INT, &regs, &regs, &sregs );
  return lpReqHdr->wStatus;
}

/*****
/* MSCDEX_SendDeviceRequest: Send device request to device          */
/*                               driver                              */
/*-----*/
/* Input   : iCD_Drive_Letter - Number of drive to whose          */
/*                               device driver a device request     */
/*                               is to be sent. (0=A, 1=B, etc.)     */
/*                               lpReqHdr - Address of device request header */
/* Output  : Status of device after operation                      */
/*                               (s. RequestHeader.wStatus)         */
/*-----*/
/* Info: This function either calls the interrupt and strategy     */
/*        routines via MSCDEX-MUX, or else calls _CallStrategy.     */
/*        The manner in which the device driver functions are called */
/*        depends on the MSCDEX version.                             */
/*****
INT MSCDEX_SendDeviceRequest( INT          iCD_Drive_Letter,
                                LPRequestHeader lpReqHdr )
{ static INT iVersion = 0x0;

  if( iVersion == 0 )          /* In case version is not known, */
                              /* determine version */
    iVersion = MSCDEX200_GetVersion();

  if( iVersion < 0x210 )      /* with versions earlier than 2.10, */
                              /* use direct call */
    return _CallStrategy( iCD_Drive_Letter, lpReqHdr );
  return MSCDEX210_SendDeviceRequest( iCD_Drive_Letter, lpReqHdr );
}

#endif

```