

```

***** C D U T I L . P A S *****
{
  -----*
  Task      : Utilities for MSCDEX programming
              Helper, Wrapper, Interfaces
  -----*
  Author     : Michael Tischer / Bruno Jennrich
  Developed on : 04/08/1994
  Last update  : 10/10/1994
  *****
}

{$A-}                                { No word alignment of structures }

Unit CDUTIL;

Interface

Const
    HSG      = 0;
    REDBOOK = 1;

{- MSCDEX - Function codes ( Pass to AL ) -----}
MSCDEX_GET_NO_OF_DRIVE_LETTERS = $00;
MSCDEX_GET_DRIVE_DEVICE_LIST   = $01;

Type
DeviceHeader = Record { Header of a DOS device driver }
    lpNextHdr      : Pointer;          { Address of next device driver }
    wDevAttributes, : Word;             { Driver attributes }
    wStrategy,      : Word;             { Offset of strategy routine }
    wInterrupt      : Word;             { Offset of interrupt routine }
    bName           : Array[0..7] of Byte; { Name of driver }
    wReserved       : Word;
    bDriveLetter,   : Word;             { Drive letter }
    bNumberOfSubUnits : Byte;           { Number of subunits }
End;

DevHeadPtr = ^DeviceHeader;           { Pointer to a driver header }

DevElement = Record
    bSubUnit        : Byte;             { Number of SubUnit }
    lpDeviceHeader   : DevHeadPtr;      { Address of device header }
End;

{- Request-Header -----}
RequestHeader = Record
    bLength,         : Word;            { Length of request structure }
    bSubUnit,        : Word;            { Number of subunit }
    bCommand : Byte;   : Word;            { Command to be executed }
    wStatus  : Word;   : Word;            { Error status }
    bReserved : Array[0..7] of Byte;    { Depending on the command }
End;                                     { will be redefined }

{ The IOCTL_READ command is used to prompt for diverse }
{ status information about the driver, drive and inserted CD. }
MSCDEX_IOCTLIO = Record
    ReqHdr      : RequestHeader;
    bMediaDescriptor : Byte;          { always 0 }
    lpTransferAddress : Pointer;       { = Address of control block }
    wTransferCount,   : Word;          { = Size of control block }
    wStartingSectorNo : Word;          { always 0 }
    lpVolumeID        : Pointer;      { always NIL }
End;

Const
{MSCDEX - Functions }
MSCDEX_GET_COPYRIGHT_FILENAME = $02;
MSCDEX_GET_ABSTRACT_FILENAME  = $03;
MSCDEX_GET_BIB_DOC_FILENAME   = $04;
MSCDEX_READ_VTOC               = $05;
MSCDEX_DEBUG_ON                = $06;
MSCDEX_DEBUG_OFF               = $07;
MSCDEX_ABSOLUTE_DISK_READ      = $08;
MSCDEX_ABSOLUTE_DISK_WRITE     = $09;
MSCDEX_RESERVED                = $0A;

{MSCDEX - Functions beginning with V2.00 }
MSCDEX200_CD_ROM_DRIVE_CHECK   = $0B;
MSCDEX200_GET_VERSION          = $0C;
MSCDEX200_GET_DRIVE_LETTERS    = $0D;
MSCDEX200_GETSET_VD_PREFERENCE = $0E;
MSCDEX200_GET_DIRECTORY_ENTRY  = $0F;
DIRECT_COPY = FALSE;
STRUCT_COPY = TRUE;

{ MSCDEX - Functions beginning with V2.10 }

```

```

MSCDEX210_SEND_DEVICE_REQUEST = $10;

{ Number of Multiplex Interrupt }
DOS_MULTIPLEX_INT = $2F;

{ ID code of MSCDEX-MUX calls }
MSCDEX_MULTIPLEX_CODE = ($1500);

iVersion : Integer = 0;
iNumUnits : Integer = -1;

Var

Strategy : Pointer; { Function addresses }
Interruptproc : Pointer;
S, O, i, iStart : Integer; { Segment, Offset }

cCDLetters : Array[0..25] of byte; { All CD drive letters }
DevElements : Array[0..25] of DevElement;

Function Time2Frame( iMin, iSec, iFrame : Integer ) : Longint;

Function Time2HSG( iMin, iSec, iFrame : Integer ) : Longint;

Function Time2REDBOOK( iMin, iSec, iFrame : Integer ) : Longint;

Procedure Frame2Time( lFrame : Longint;
var Min, Sec, Frame : Integer );

Procedure REDBOOK2Time( lREDBOOK : Longint;
var Min, Sec, Frame : Integer );

Procedure HSG2Time( lHSG : Longint;
var Min, Sec, Frame : Integer );

Function REDBOOK2HSG( lREDBOOK : Longint ) : Longint;

Function HSG2REDBOOK( lHSG:Longint ) : Longint;

Function MSCDEX_GetNumberOfDriveLetters( var Number,
StartLetter : Integer ) : Boolean;

Function MSCDEX_Installed : Boolean;

Function MSCDEX_GetCDRomDriveDeviceList( DevElArray : pointer ) : Boolean;

Function MSCDEX_GetCopyrightFilename( iCD_Drive_Letter : Integer;
lpName : Pointer ) : Boolean;

Function MSCDEX_GetAbstractFilename( iCD_Drive_Letter : Integer;
lpName : Pointer ) : Boolean;

Function MSCDEX_GetBibDocFilename( iCD_Drive_Letter : Integer;
lpName : Pointer ) : Boolean;

Function MSCDEX_ReadVTOC( iCD_Drive_Letter : Integer;
lpSector : Pointer;
iNumSec : Integer ) : Integer;

Function MSCDEX_DebugOn : Boolean;

Function MSCDEX_DebugOff : Boolean;

Function MSCDEX_AbsoluteRead( iCD_Drive_Letter : Integer;
lpSector : Pointer;
iSecCnt : Integer;
lSecStart : Longint ) : Boolean;

Function MSCDEX_AbsoluteWrite( iCD_Drive_Letter : Integer;
lpSector : Pointer;
iSecCnt : Integer;
lSecStart : Longint ) : Boolean;

Function MSCDEX200_CDRomDriveCheck( iCD_Drive_Letter : Integer ) : Boolean;

Function MSCDEX200_GetVersion : Integer;

Function MSCDEX200_GetCDRomDriveLetters( lpLetters : Pointer ) : Boolean;

Function MSCDEX200_GetVDPPreference( iCD_Drive_Letter : Integer;
var Preference : Integer ) : Boolean;

Function MSCDEX200_SetVDPPreference( iCD_Drive_Letter,
iPreference : Integer ) : Boolean;

```

```

Function MSCDEX200_GetDirectoryEntry(      iCD_Drive_Letter : Integer;
                                           iCopyFlag       : Boolean;
                                           PathName        : String;
                                           lpData          : Pointer;
                                           var HSG          : Integer ) : Boolean;

Function MSCDEX210_SendDeviceRequest(      iCD_Drive_Letter : Integer;
                                           var ReqHdr        : RequestHeader ) : Integer;

Function _CallStrategy(      iCD_Drive_Letter : Integer;
                             var ReqHdr        : RequestHeader ) : Integer;

Function MSCDEX_SendDeviceRequest(      iCD_Drive_Letter : Integer;
                                         var ReqHdr        : RequestHeader ) : Integer;

Function MSCDEX_ReadWriteReq( iCD_Drive_Letter,
                              iCommand : Integer;
                              lpControlBlock : Pointer;
                              iControlBlockSize : Integer ) : Integer;

Implementation
Uses DOS;

{*****}
{ 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 }
{*****}
Function Time2Frame( iMin, iSec, iFrame : Integer ) : Longint;

Begin
    Time2Frame := iMin * LongInt( 60 * 75 ) +
                  iSec * LongInt( 75 ) +
                  iFrame;
End;

{*****}
{ Time2HSG : Converts time to HSG address }
{*****}
{ Input   : iMin   - Minute ( 0 - 59+ ) }
{          iSec   - Second ( 0 - 59 ) }
{          iFrame - Frame ( 0 - 75 ) }
{ Output  : Converted HSG address }
{*****}
Function Time2HSG( iMin, iSec, iFrame : Integer ) : Longint;

Begin
    Time2HSG := Time2Frame( iMin, iSec, iFrame ) - 150;
End;

{*****}
{ Time2REDBOOK : Converts time to REDBOOK address }
{*****}
{ Input   : iMin   - Minute ( 0 - 59+ ) }
{          iSec   - Second ( 0 - 59 ) }
{          iFrame - Frame ( 0 - 75 ) }
{ Output  : Converted REDBOOK address }
{*****}
Function Time2REDBOOK( iMin, iSec, iFrame : Integer ) : Longint;

Begin
    Time2REDBOOK := ( ( LongInt( iMin )   shl 16 ) or
                      ( LongInt( iSec )   shl  8 ) or
                      ( LongInt( iFrame )  shl  0 ) );
End;

{*****}
{ FrameTime : Splits number of frames (sectors) into minute, }
{             second, frame. }
{*****}
{ Input   : lFrames - number of frames (sectors) to be converted }
{          Min     - Integer variable for receiving minutes }
{          Sec     - ditto for seconds }
{          Frame   - ditto for frames }
{*****}
Procedure Frame2Time(      lFrame          : Longint;
                          var Min, Sec, Frame : Integer );

Begin
    Min := Integer ( lFrame div ( LongInt( 60 * 75 ) ) );
    lFrame := lFrame - Min * LongInt( 60 * 75 );
    Sec := Integer ( lFrame div LongInt( 75 ) );
    lFrame := lFrame - Sec * LongInt( 75 );

```

```

Frame := Integer ( lFrame );
End;

{*****}
{ REDBOOK2Time : Splits REDBOOK address into minute, second, frame. }
{*****}
{ Input      : lREDBOOK - REDBOOK address to be converted }
{           Min      - Integer variable for receiving the minutes }
{           Sec      - ditto for seconds }
{           Frame    - ditto for frames }
{*****}
Procedure REDBOOK2Time(      lREDBOOK      : Longint;
                          var Min, Sec, Frame : Integer );
Begin
  Frame := Integer ( lREDBOOK shr 0 ) and $00FF;
  Sec   := Integer ( lREDBOOK shr 8 ) and $00FF;
  Min   := Integer ( lREDBOOK shr 16 ) and $00FF;
End;

{*****}
{ HSG2Time : Splits HSG address into minute, second, frame. }
{*****}
{ Input      : lHSG - HSG address to be converted }
{           Min      - Integer variable for receiving the minutes }
{           Sec      - ditto for seconds }
{           Frame    - ditto for frames }
{*****}
Procedure HSG2Time(      lHSG      : Longint;
                      var Min, Sec, Frame : Integer );
Begin
  Frame2Time( lHSG + LongInt( 150 ), Min, Sec, Frame );
End;

{*****}
{ REDBOOK2HSG : Convert REDBOOK address to HSG address }
{*****}
{ Input      : lREDBOOK - REDBOOK address to be converted }
{ Output     : converted HSG address }
{*****}
Function REDBOOK2HSG( lREDBOOK : Longint ) : Longint;

var iMin, iSec, iFrame : Integer;

Begin
  REDBOOK2Time( lREDBOOK, iMin, iSec, iFrame );
  REDBOOK2HSG := Time2HSG( iMin, iSec, iFrame );
End;

{*****}
{ HSG2REDBOOK : Convert HSG address to REDBOOK address }
{*****}
{ Input      : lHSG - HSG address to be converted }
{ Output     : converted REDBOOK address }
{*****}
Function HSG2REDBOOK( lHSG : Longint ):Longint;

var iMin, iSec, iFrame : Integer;

Begin
  HSG2Time( lHSG, iMin, iSec, iFrame );
  HSG2REDBOOK := Time2REDBOOK( iMin, iSec, iFrame );
End;

{- MSCDEX Interface Functions -----}

{*****}
{ MSCDEX_GetNumberOfDriveLetters : Get number of supported }
{                               CD-ROM drives. }
{*****}
{ Input      : Number      - Integer variable that is to receive the }
{                               number of drives (output parameter) }
{           StartLetter - ASCII code of first drive letter to }
{                               be used by a CD-ROM }
{                               drive. }
{ Output     : TRUE - Operation successful }
{           FALSE - Operation failed }
{*****}
{ 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. }
{*****}
Function MSCDEX_GetNumberOfDriveLetters( var Number,
                                         StartLetter : Integer ) : Boolean;

var regs:Registers;

```

```

Begin
    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX_GET_NO_OF_DRIVE_LETTERS;
    regs.bx := 0;
    intr( DOS_MULTIPLEX_INT, regs );
    if ( regs.flags and 1 ) = 0 then
        { Carry flag set? }
    Begin
        Number := regs.bx;
        StartLetter := regs.cx;
        MSCDEX_GetNumberOfDriveLetters := TRUE;
    End
    else Begin
        Number := 0;
        StartLetter := 0;
        MSCDEX_GetNumberOfDriveLetters := FALSE;
    End;
End;

{ ***** }
{ MSCDEX_Installed : MSCDEX installed ? }
{ ***** }
{ *-----* }
{ Info : This function tests whether the number of supported }
{ drives differs from 0. }
{ ***** }
Function MSCDEX_Installed : Boolean;
var iNumber, iStartLetter : Integer;
    dummy : boolean;

Begin
    iNumber := 0;
    dummy := MSCDEX_GetNumberOfDriveLetters( iNumber, iStartLetter );
    if iNumber = 0 then MSCDEX_Installed := FALSE
        else MSCDEX_Installed := TRUE;
End;

{ ***** }
{ MSCDEX_GetCDRomDriveDeviceList : Get information about }
{ connected CD-ROM drives. }
{ ***** }
{ *-----* }
{ Input : DevElArray - Pointer to array with elements of the }
{ DevElement type in which the information }
{ about the connected drives is entered. }
{ 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. }
{ ***** }
Function MSCDEX_GetCDRomDriveDeviceList( DevElArray : pointer ) : Boolean;

var regs : Registers;

Begin
    { AX = $1501 }
    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX_GET_DRIVE_DEVICE_LIST ;
    regs.es := Word ( Longint ( DevElArray ) shr 16 );
    regs.bx := Word ( Longint ( DevElArray ) and $FFFF );
    intr( DOS_MULTIPLEX_INT, regs);
    if ( regs.flags and 1 ) = 0 then MSCDEX_GetCDRomDriveDeviceList := TRUE
        else MSCDEX_GetCDRomDriveDeviceList := FALSE;
End;

{ ***** }
{ 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. }
{ ***** }
Function MSCDEX_GetCopyrightFilename( iCD_Drive_Letter : Integer;
    lpName : Pointer ) : Boolean;

var regs : Registers;

Begin
    { AX = $1502, CX = Drive letter }
    { ES:BX = Address of name }

```

```

regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX_GET_COPYRIGHT_FILENAME;
regs.es := Longint ( lpName ) shr 16;
regs.bx := Longint ( lpName ) and $ffff;
regs.cx := iCD_Drive_Letter;
intr( DOS_MULTIPLEX_INT, regs);
if ( regs.flags and 1 ) = 0 then MSCDEX_GetCopyrightFilename := TRUE
                                else MSCDEX_GetCopyrightFilename := FALSE;
End;

{*****}
{ 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.
}
{*****}
Function MSCDEX_GetAbstractFilename( iCD_Drive_Letter : Integer;
                                    lpName           : Pointer ) : Boolean;

var regs : Registers;

Begin
    { AX = $1503, CX = Drive letter }
    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX_GET_ABSTRACT_FILENAME;
    regs.es := Longint ( lpName ) shr 16;
    regs.bx := Longint ( lpName ) and $ffff;
    regs.cx := iCD_Drive_Letter;
    intr( DOS_MULTIPLEX_INT, regs);
    if ( regs.flags and 1 ) = 0 then MSCDEX_GetAbstractFilename := TRUE
                                    else MSCDEX_GetAbstractFilename := FALSE;
End;

{*****}
{ 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.
}
{*****}
Function MSCDEX_GetBibDocFilename( iCD_Drive_Letter : Integer;
                                   lpName           : Pointer ) : Boolean;

var regs : Registers;

Begin
    { AX = $1504, CX = Drive letter }
    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX_GET_BIB_DOC_FILENAME ;
    regs.es := Longint ( lpName ) shr 16;
    regs.bx := Longint ( lpName ) and $ffff;
    regs.cx := iCD_Drive_Letter;
    intr( DOS_MULTIPLEX_INT, regs );
    if ( regs.flags and 1 ) = 0 then MSCDEX_GetBibDocFilename := TRUE
                                    else MSCDEX_GetBibDocFilename := FALSE;
End;

{*****}
{ 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 TOC sector to be read.
}
{ Output  : <> 0 - Operation successful
            $0101 - Read sector = standard VTOC sector
            $01FF - Read sector = last VTOC sector
            $0100 - Read sector = other VTOC-sector
            0      - Error (s. _doserrno )
}
{*****}
Function MSCDEX_ReadVTOC( iCD_Drive_Letter : Integer;
                         lpSector         : Pointer;
                         iNumSec          : Integer ) : Integer;

var regs : Registers;

Begin
    { AX = $1505, CX = Drive letter
      ES:BX = Address of name }
    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX_READ_VTOC;
    regs.es := Longint ( lpSector ) shr 16;
    regs.bx := Longint ( lpSector ) and $ffff;

```

```

regs.cx := iCD_Drive_Letter;
regs.dx := iNumSec;
intr( DOS_MULTIPLEX_INT, regs );
if ( regs.flags and 1 ) = 0 then MSCDEX_ReadVTOC := ( $0100 or regs.al )
    else MSCDEX_ReadVTOC := 0;
End;

{*****}
{ MSCDEX_DebugOn : Enable debugging.
  (only for MSCDEX debug version)
}
{-----*}
{ Output      : TRUE  - Operation successful
  FALSE - Operation failed ( Error in _doserrno )
}
{*****}
Function MSCDEX_DebugOn : Boolean;

var regs : Registers;

Begin
    { AX = $1506 }
    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX_DEBUG_ON;
    intr( DOS_MULTIPLEX_INT, regs );
    if ( regs.flags and 1 ) = 0 then MSCDEX_DebugOn := TRUE
        else MSCDEX_DebugOn := FALSE;
End;

{*****}
{ MSCDEX_DebugOff : Disable debugging.
  (only for MSCDEX debug version)
}
{-----*}
{ Output      : TRUE  - Operation successful
  FALSE - Operation failed ( Error in _doserrno )
}
{*****}
Function MSCDEX_DebugOff : Boolean;

var regs : Registers;

Begin
    { AX = $1507 }
    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX_DEBUG_OFF;
    intr( DOS_MULTIPLEX_INT, regs );
    if ( regs.flags and 1 ) = 0 then MSCDEX_DebugOff := TRUE
        else MSCDEX_DebugOff := FALSE;
End;

{*****}
{ 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.
}
{*****}
Function MSCDEX_AbsoluteRead( iCD_Drive_Letter : Integer;
    lpSector      : Pointer;
    iSecCnt       : Integer;
    lSecStart     : Longint ) : Boolean;

var regs : Registers;

Begin
    { AX = $1508, CX = Drive letter
      DX = Number of sectors to be read
      ES:BX = Address of the read buffer
      SI:DI = Address of the first sector
    }

    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX_ABSOLUTE_DISK_READ ;
    regs.es := Longint ( lpSector ) shr 16;
    regs.bx := Longint ( lpSector ) and $FFFF;
    regs.cx := iCD_Drive_Letter;
    regs.dx := iSecCnt;
    regs.si := lSecStart shr 16;
    regs.di := lSecStart and $ffff;
    intr( DOS_MULTIPLEX_INT, regs );
    if ( regs.flags and 1 ) = 0 then MSCDEX_AbsoluteRead := TRUE
        else MSCDEX_AbsoluteRead := FALSE;
End;

{*****}
{ 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.
}
*****}
Function MSCDEX_AbsoluteWrite( iCD_Drive_Letter : Integer;
                               lpSector : Pointer;
                               iSecCnt : Integer;
                               lSecStart : Longint ) : Boolean;

var regs : Registers;

Begin
    { AX = $1509, CX = Drive letter
      DX = Number of sectors to be written
      ES:BX = Address of write buffer
      SI:DI = Address of first sector
    }

    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX_ABSOLUTE_DISK_WRITE;
    regs.es := Longint ( lpSector ) shr 16;
    regs.bx := Longint ( lpSector ) and $ffff;
    regs.cx := iCD_Drive_Letter;
    regs.dx := iSecCnt;
    regs.si := lSecStart shr 16;
    regs.di := lSecStart and $ffff;
    intr( DOS_MULTIPLEX_INT, regs );
    if ( regs.flags and 1 ) = 0 then MSCDEX_AbsoluteWrite := TRUE
    else MSCDEX_AbsoluteWrite := FALSE;

End;

{ *****
  { 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 )
  }
  *****}
Function MSCDEX200_CDROMDriveCheck( iCD_Drive_Letter : Integer ) : Boolean;

var regs : Registers;

Begin
    { AX = $150B, CX = Drive letter }

    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX200_CD_ROM_DRIVE_CHECK;
    regs.cx := iCD_Drive_Letter;
    regs.bx := 0;
    intr( DOS_MULTIPLEX_INT, regs );
    if ( ( regs.bx = $ADAD ) and ( regs.ax <> 0 ) ) then
        MSCDEX200_CDROMDriveCheck := TRUE else
        MSCDEX200_CDROMDriveCheck := FALSE;

End;

{ *****
  { 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!
  }
  *****}
Function MSCDEX200_GetVersion : Integer;

var regs : Registers;

Begin
    { AX = $150C }
    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX200_GET_VERSION;
    regs.bx := 0;
    { Clear first BX }
    intr( DOS_MULTIPLEX_INT, regs );
    if ( ( regs.bx <> 0 ) and ( ( regs.flags and 1 ) = 0 ) ) then
        MSCDEX200_GetVersion:=regs.bx
    else
        if ( regs.flags and 1 ) = 0 then
            MSCDEX200_GetVersion:=$0100
        else
            MSCDEX200_GetVersion := -1;
        { V1.00 or error }
    end if;

End;

{ *****
  { 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 )
    }
    {*****}
Function MSCDEX200_GetCDRomDriveLetters( lpLetters : Pointer ) : Boolean;

var regs : Registers;

Begin
    { AX = $150D }
    { ES:BX = Adresse des Arrays }

    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX200_GET_DRIVE_LETTERS;
    regs.es := Longint ( lpLetters ) shr 16;
    regs.bx := Longint ( lpLetters ) and $ffff;
    intr( DOS_MULTIPLEX_INT, regs );
    if ( regs.flags and 1 ) = 0 then MSCDEX200_GetCDRomDriveLetters := TRUE
    else MSCDEX200_GetCDRomDriveLetters := FALSE;

End;

{*****}
{ 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 )
}
{*****}
Function MSCDEX200_GetVDPPreference( iCD_Drive_Letter : Integer;
var Preference : Integer ) : Boolean;

var regs : Registers;

Begin
    { AX = $150D }

    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX200_GETSET_VD_PREFERENCE;
    regs.cx := iCD_Drive_Letter;
    regs.bx := $0000; { 0 = Get Preference }
    regs.dx := 0; { Clear result register as precaution }
    intr( DOS_MULTIPLEX_INT, regs );
    Preference := regs.dx;
    if ( regs.flags and 1 ) = 0 then MSCDEX200_GetVDPPreference := TRUE
    else MSCDEX200_GetVDPPreference := FALSE;

End;

{*****}
{ 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 )
}
{*****}
Function MSCDEX200_SetVDPPreference( iCD_Drive_Letter,
iPreference : Integer ) : Boolean;

var regs : Registers;

Begin
    { AX = $150D }

    regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX200_GETSET_VD_PREFERENCE;
    regs.cx := iCD_Drive_Letter;
    regs.bx := $0001; { 1 = Set Preference }
    regs.dx := iPreference;
    intr( DOS_MULTIPLEX_INT, regs );
    if ( regs.flags and 1 ) = 0 then MSCDEX200_SetVDPPreference := TRUE
    else MSCDEX200_SetVDPPreference := FALSE;

End;

{*****}
{ 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
  PathName - Path name of file
  lpData - Address to which data are to be
  copied
  HSG - 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 variable passed for the HSG
  parameter indicates which of the two types
  (HSG_ENTRY or ISO_ENTRY) you are dealing with.
}

```

```

{*****}
Function MSCDEX200_GetDirectoryEntry(      iCD_Drive_Letter : Integer;
                                           iCopyFlag       : Boolean;
                                           PathName        : String;
                                           lpData          : Pointer;
                                           var HSG          : Integer ) : Boolean;

var regs : Registers;

Begin
    { AX = $150F, CL = CD_Drive, CH = copy flags }
    { ES:BX = Pathname, SI:DI = Data address }

regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX200_GET_DIRECTORY_ENTRY;
regs.cl := Byte ( iCD_Drive_Letter );

    { Set/clear Bit 0 }
if iCopyFlag then regs.ch := $01 else regs.ch := $00;
Pathname := Pathname+#0;
regs.es := Longint ( @PathName ) shr 16;
regs.bx := Longint ( @PathName ) and $ffff + 1;
regs.si := Longint ( lpData ) shr 16;
regs.di := Longint ( lpData ) and $ffff;
intr( DOS_MULTIPLEX_INT, regs );
HSG := regs.ax;
if ( regs.flags and 1 ) = 0 then MSCDEX200_GetDirectoryEntry := TRUE
    else MSCDEX200_GetDirectoryEntry := FALSE;

End;

{*****}
{ 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. }
{*****}
Function MSCDEX210_SendDeviceRequest(      iCD_Drive_Letter : Integer;
                                           var ReqHdr        : RequestHeader ) : Integer;

var regs : Registers;

Begin
    { AX = $1510, CX = Drive letter }
    { ES:BX = Address of request header }

regs.ax := MSCDEX_MULTIPLEX_CODE or MSCDEX210_SEND_DEVICE_REQUEST;
regs.cx := iCD_Drive_Letter;
regs.es := Longint ( @ReqHdr ) shr 16;
regs.bx := Longint ( @ReqHdr ) and $ffff;
intr( DOS_MULTIPLEX_INT, regs );
MSCDEX210_SendDeviceRequest := ReqHdr.wStatus;

End;

{*****}
{ _CallStrategy: 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.) }
{          ReqHdr          - The device request header }
{ Output  : Status of device after operation (s. RequestHeader.wStatus) }
{-----*-----}
{ Info: This function calls the strategy first, then the interrupt function of the device driver whose header and SubUnit are determined by the passed drive letters. This function is called when the version number of MSCDEX is < 2.10. }
{*****}
Function _CallStrategy(      iCD_Drive_Letter : Integer;
                             var ReqHdr        : RequestHeader ) : Integer;

var lpDevHdr : DevHeadPtr;
    dummy     : boolean;

Begin
if iNumUnits = -1 then { All static variables initialized ? }
Begin
    dummy := MSCDEX_GetNumberOfDriveLetters( iNumUnits, iStart );
    dummy := MSCDEX_GetCDRomDriveDeviceList( @DevElements[0] );

if iVersion >= $200 then { Does GetCDRomDriveLetters() exist? }
    dummy := MSCDEX200_GetCDRomDriveLetters( @cCDLetters )
else
    { otherwise sequential drive letters }
    for i := 0 to iNumUnits - 1 do

```

```

cCDLetters[i] := Byte( iStart + i );
End;

for i := 0 to iNumUnits - 1 do
  if ( cCDLetters[i] = iCD_Drive_Letter ) then      { Which drive? }
  Begin
    lpDevHdr := DevElements[i].lpDeviceHeader; { DeviceHeader and }
    ReqHdr.bSubUnit := DevElements[i].bSubUnit; { SubUnit }
    if ( Longint ( lpDevHdr ) and $ffff ) <> 0 then
    Begin
      _CallStrategy := -1; { Always to Offset 0 }
      Exit;
    End;
    Strategy := Pointer ( ( Longint ( lpDevHdr ) and $ffff0000 )
      + lpDevHdr^.wStrategy );
    Interruptproc := Pointer ( ( Longint ( lpDevHdr ) and $ffff0000 )
      + lpDevHdr^.wInterrupt ); { Get Segment and Offset before }
    S := Longint ( @ReqHdr ) shr 16; { running asm instructions! }
    O := Longint ( @ReqHdr ) and $ffff;
    asm
      mov es,S { Set ES:BX }
      mov bx,O
      call Strategy { First set internal buffer addresses... }
      call Interruptproc; { ...then run code }
    End;
    _CallStrategy := ReqHdr.wStatus;
    Exit;
  End;
  _CallStrategy := -1;
End;

{*****}
{ 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.) }
{ ReqHdr - Device request header }
{ Output : Status of device after operation }
{ (s. RequestHeader.wStatus) }
{-----*}
{ Info: This function calls the interrupt and strategy routines }
{ either via the MSCDEX-MUX, or by calling _CallStrategy. }
{ The manner in which the device driver functions are called }
{ depends on the MSCDEX version. }
{*****}
Function MSCDEX_SendDeviceRequest( iCD_Drive_Letter : Integer;
  var ReqHdr : RequestHeader ) : Integer;
Begin
  if iVersion = 0 then { If version not known, get version }
    iVersion := MSCDEX200_GetVersion;

  if iVersion < $210 then { With versions lower than 2.10 direct call }
    MSCDEX_SendDeviceRequest :=
      _CallStrategy( iCD_Drive_Letter, ReqHdr )
  else
    MSCDEX_SendDeviceRequest :=
      MSCDEX210_SendDeviceRequest( iCD_Drive_Letter, ReqHdr );
End;

{*****}
{ 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 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()) }
{*****}
Function MSCDEX_ReadWriteReq( iCD_Drive_Letter,
  iCommand : Integer;
  lpControlBlock : Pointer;
  iControlBlockSize : Integer ) : Integer;

var IOCTLIO : MSCDEX_IOCTLIO;

```

```
Begin
    IOCTLIO.RegHdr.bLength      := sizeof( RequestHeader );
    IOCTLIO.RegHdr.bSubUnit     := 0;
    IOCTLIO.RegHdr.bCommand      := Byte ( iCommand );
    IOCTLIO.bMediaDescriptor     := 0;
    IOCTLIO.lpTransferAdress     := lpControlBlock;
    IOCTLIO.wTransferCount       := iControlBlockSize;
    IOCTLIO.wStartingSectorNo    := 0;
    IOCTLIO.lpVolumeID           := NIL;

    MSCDEX_ReadWriteReq:=MSCDEX_SendDeviceRequest( iCD_Drive_Letter,
                                                    IOCTLIO.RegHdr );
End;

End.
```