

Listing: NETFILEC.C

```

/*****
/*                                N E T F I L E C . C                                */
/*-----*/
/*   Task           : Implements network supporting file functions.           */
/*-----*/
/*   Memory model   : SMALL                                                    */
/*-----*/
/*   Author          : Michael Tischer                                         */
/*   developed on    : 02/10/1992                                              */
/*   last Update     : 04/07/1995                                              */
/*-----*/
/*   Microsoft C     : The "Segment lost in conversation" warning,           */
/*                     cannot be avoided.                                       */
*****/

```

```

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

```

```

/*== Macros =====*/

```

```

#ifdef FP_OFF
    #undef FP_OFF
    #undef FP_SEG
#endif

```

```

#define FP_OFF(fp)      ((unsigned)(fp))
#define FP_SEG(fp)      ((unsigned)((unsigned long)(fp) >> 16))

```

```

/*== Constants =====*/

```

```

/*-- Wahrheitswerte -----*/

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

/*-- Types of file access -----*/

#define FM_R      0                /* read only */
#define FM_W      1                /* write only */
#define FM_RW     2                /* read and write */

/*-- Types of file protection -----*/

#define SM_COMP 0x00              /* compatibility mode, no file protection */
#define SM_RW   0x10              /* read and write prohibited by others */
#define SM_R    0x20 /* read by others permitted, writting prohibited */
#define SM_W    0x30 /* reading and writing by other permitted */
#define SM_NO   0x40              /* all permitted, protected by Record Lock */

/*-- Possible errors during procedure calls -----*/

#define NE_OK          0x00                /* no error */
#define NE_FileNotFound 0x02                /* error: file not found */
#define NE_PathNotFound 0x03                /* error: Path not found */
#define NE_TooManyFiles 0x04                /* error: to many open files */
#define NE_AccessDenied 0x05                /* error: access to file denied */
#define NE_InvalidHandle 0x06              /* error: invalid file handle */
#define NE_AccessCode   0x07              /* error: illegal access code */
#define NE_Share         0x20              /* violation of share error */
#define NE_Lock          0x21              /* error: (Un)locking of a record */
#define NE_ShareBuffer   0x24              /* Share buffer overflow */

```

```

/*-- Function numbers for DOS calls -----*/

#define FCT_OPEN      0x3D      /* Function: open file with handle */
#define FCT_CLOSE     0x3E      /* Function: close file with handle */
#define FCT_CREATE    0x3C      /* Function: create file with handle */
#define FCT_WRITE     0x40      /* Function: write to file */
#define FCT_READ      0x3F      /* Function: read from file */
#define FCT_LSEEK     0x42      /* Function: set file pointer */
#define FCT_REC_LOCK  0x5C      /* Function: Record Locking */

/*-- Function and interrupt numbers for other interrupt calls -----*/

#define MULTIPLEX      0x2F      /* Multiplex Interrupt */
#define FCT_SHARE      0x1000    /* Test if share is installed */

/*-- File indentifiers (Werte analog Turbo-Pascal) -----*/

#define FMCLOSED       0xD7B0    /* file closed */
#define FMINPUT        0xD7B1    /* file opened for reading */
#define FMOUTPUT       0xD7B2    /* file opened for writing */
#define FMINOUT        0xD7B3    /* file opened for reading and writing */

/*== Type declarations =====*/

typedef struct { unsigned int Handle, RecS, Mode; } NFILE;

/*== Global Variables =====*/

int      NetError;              /* error number from DOS Interrupt */
union REGS regs;                /* processor registers for interrupt call */
struct SREGS sregs;             /* segment register for Interrupt call */

```

```

/*****
/* ShareInst      : Test if Share has been installed      */
/* Input          : none                                   */
/* Ouput          : TRUE if Share is installed             */
/* Globale Var.   : NetError/W (error status from call)   */
*****/

```

```

int ShareInst( void )
{
    regs.x.ax = FCT_SHARE;                /* test if Share is installed */
    int86( MULTIPLEX, &regs, &regs );    /* multiplex Interrupt call */
    NetError = NE_OK;                    /* no error */
    return ( regs.h.al == 0xFF );        /* return result */
}

```

```

/*****
/* NetErrorMsg    : Error message text                    */
/* Input          : s.u.                                   */
/* Output         : s.u.                                   */
*****/

```

```

void NetErrorMsg( int   Number,          /* error number */
                  char *Text )          /* error text */
{
    char Sdummy[ 5 ];                  /* error number */

    switch ( Number )
    {
        case NE_OK                : strcpy( Text, "No error"           " );
                                   break;
        case NE_FileNotFound       : strcpy( Text, "File not found"     " );
                                   break;
    }
}

```

```

case NE_PathNotFound : strcpy( Text, "Path not found           " );
                        break;
case NE_TooManyFiles : strcpy( Text, "Too many open files    " );
                        break;
case NE_AccessDenied : strcpy( Text, "File access denied     " );
                        break;
case NE_InvalidHandle: strcpy( Text, "Invalid file handle    " );
                        break;
case NE_AccessCode   : strcpy( Text, "Illegal access code    " );
                        break;
case NE_Share         : strcpy( Text, "Violation of Share rights " );
                        break;
case NE_Lock          : strcpy( Text, "Error during record lock " );
                        break;
case NE_ShareBuffer   : strcpy( Text, "Share buffer overflow   " );
                        break;
default               : {
                        itoa( Number, Sdummy, 2 );
                        strcpy( Text, "DOS Error:                " );
                        strcat( Text, Sdummy );
                        }
}
}

```

```

/*****
/* NetReset      : opens a specific file          */
/* Input         : s.u.                          */
/* Output        : s.u.                          */
/* Globale Var.: NetError/W (error status from call) */
*****/

```

```

void NetReset( char far      *DName,                      /* file name */

```

```

        unsigned int AMode,                /* access mode */
        unsigned int RecS,                 /* record size */
        NFILE          *TFile )           /* pointer to file */
{
    regs.x.dx = FP_OFF( DName );           /* assign filename */
    regs.h.ah = FCT_OPEN;                  /* function number: Open file */
    regs.h.al = AMode;                     /* status byte for access mode and locking */
    sregs.ds = FP_SEG( DName );
    intdosx( &regs, &sregs );             /* call interrupt */
    if ( !regs.x.cflag )                   /* open successful? */
    {
        TFile->Handle = regs.x.ax;         /* specify file handle */
        TFile->RecS = RecS;                /* specify record size */
        switch ( AMode & 0x0F )           /* set access mode */
        {
            case FM_R : TFile->Mode = FMINPUT;
                        break;
            case FM_W : TFile->Mode = FMOUTPUT;
                        break;
            case FM_RW : TFile->Mode = FMINOUT;
                        break;
        }
        NetError = NE_OK;                 /* no error */
    }
    else
        NetError = regs.x.ax;             /* note error number */
}

/*****
/* NetRewrite : Creates a file
/* Input : s.u.
/* Output : s.u

```

```

/* Globale Var.: NetError/W (error status from call) */
/*****

void NetRewrite( char far      *DName,          /* filename */
                unsigned int  AMode,          /* access mode */
                unsigned int  RecS,          /* Record size */
                NFILE         *TFile )       /* pointer to file */
{
    regs.x.dx = FP_OFF( DName );              /* Adresse des Dateinamens */
    regs.h.ah = FCT_CREATE;                   /* function number: Open file */
    regs.x.cx = 0 ;                          /* file attribute */
    sregs.ds = FP_SEG( DName );
    intdosx( &regs, &regs, &sregs );         /* call interrupt */
    if ( !regs.x.cflag )                      /* open successful? */
    {
        regs.x.bx = regs.x.ax;               /* Handle in Register BX */
        regs.h.ah = FCT_CLOSE;               /* Function number close file */
        intdos( &regs, &regs );
        if ( !regs.x.cflag )                 /* close successful? */
            NetReset( DName, AMode, RecS, TFile ); /* open file */
        else
            NetError = regs.x.ax;            /* note error number */
    }
    else
        NetError = regs.x.ax;                /* note error number */
}

/*****
/* NetClose : close a file */
/* Input : s.u. */
/* Output : none */
/*****/

```

```

void NetClose( NFILE *TFile )
{
    if ( TFile->Mode != FMCLOSED )                /* file open? */
    {
        regs.x.bx = TFile->Handle;                /* file handle */
        regs.h.ah = FCT_CLOSE;                    /* function number: Close file */
        intdos( &regs, &regs );
        if ( !regs.x.cflag )                      /* file closed? */
        {
            TFile->Handle = 0;                    /* reset handle */
            TFile->Mode = FMCLOSED;                /* close file */
            NetError = NE_OK;                      /* no error */
        }
        else
            NetError = regs.x.ax;
    }
    else
        NetError = NE_InvalidHandle;              /* file not open */
}

/*****
/* Locking      : Locks or unlocks a file range.          */
/* Input        : s.u                                     */
/* Output        : TRUE is successful                      */
/* Globale Var.: NetError/W (error status from call)      */
/* Info         : Call NetLock and NetUnlock for internal access only */
*****/

int Locking( int          Handle,                /* file handle */
             int          Operation,             /* Oper. Lock, Unlock */
             unsigned long Offset, /* Offset zum Dateianfang in Bytes */

```



```

        unsigned long WrdLen )    /* L_nge des Bereiches in Bytes */
{
    regs.h.ah = FCT_REC_LOCK;      /* function number for interrupt call */
    regs.h.al = Operation;          /* 0 = Lock, 1 = Unlock */
    regs.x.bx = Handle;             /* file handle */
    regs.x.cx = Offset >> 16;      /* Hi Word Offset */
    regs.x.dx = Offset & 0xFFFF;   /* Lo Word Offset */
    regs.x.si = WrdLen >> 16;      /* Hi Word Length */
    regs.x.di = WrdLen & 0xFFFF;   /* Lo Word Length */
    intdos( &regs, &regs );        /*call Interrupt */
    if ( ! regs.x.cflag )           /* locking successful? */
    {
        NetError = NE_OK;
        return TRUE;               /* no error */
    }
    else
    {
        NetError = regs.x.ax;       /* note error number */
        return FALSE;              /* error */
    }
}

/*****
/* NetUnLock    : Unlocks locked records.
/* Input       : s.u.
/* Output      : TRUE if successful
/* Globale Var.: NetError/W (error status from call)
*****/

int NetUnLock( NFILE          *TFile,                /* file */
               unsigned long RecNo,                  /* Record Number */
               unsigned long RngNum )                /* number of Records */

```

```

{
    return Locking( TFile->Handle, 1, TFile->RecS * RecNo,
                    TFile->RecS * RngNum );
}

/*****
/* NetLock      : Locks records.                               */
/* Input        : s.u.                                         */
/* Output       : TRUE if successful                           */
/* Globale Var.: NetError/W (error status from call)          */
*****/

int NetLock( NFILE      *TFile,                                /* file */
             unsigned long RecNo,                             /* record number */
             unsigned long RngNum )                            /* number of Records */
{
    return Locking( TFile->Handle, 0, TFile->RecS * RecNo,
                    TFile->RecS * RngNum );
}

/*****
/* Is_NetReadOk : Enables file input.                           */
/* Input        : s.u.                                         */
/* Output       : TRUE if output is enabled                   */
*****/

int Is_NetReadOk( NFILE *TFile )
{
    return ( ( TFile->Mode == FMINPUT ) ||
             ( TFile->Mode == FMINOUT ) );
}

```

```

/*****
/* Is_NetOpen      : Checks the status of a file          */
/* Input           : s.u                                  */
/* Output          : TRUE if file is open                 */
*****/

```

```

int Is_NetOpen( NFILE *TFile )
{
    return ( ( TFile->Mode == FMOUTPUT ) ||
              ( TFile->Mode == FMINPUT ) ||
              ( TFile->Mode == FMINOUT ) );
}

```

```

/*****
/* Is_NetWriteOk   : Is a write to TFile allowed?        */
/* Input           : s.u                                  */
/* Output          : TRUE is output enabled              */
*****/

```

```

int Is_NetWriteOk( NFILE *TFile )
{
    return ( ( TFile->Mode == FMOUTPUT ) ||
              ( TFile->Mode == FMINOUT ) );
}

```

```

/*****
/* NetWrite        : Writes to a file.                   */
/* Input           : s.u.                                  */
/* Output          : none                                  */
*****/

```

```

void NetWrite( NFILE      *TFile,                               /* file */

```



```

NetError = regs.x.ax;                                /* note error number */
}

/*****
/* NetSeek      : Sets file pointer.
/* Input       : s.u
/* Output      : none
*****/

void NetSeek( NFILE      *TFile,                      /* file */
              unsigned long RecNo )                  /* Record number */
{
    regs.h.ah = FCT_LSEEK;                            /* function number: set file pointer */
    regs.h.al = 0;                                    /* absolute position for start of file */
    regs.x.bx = TFile->Handle;                        /* file handle */
    RecNo = RecNo * TFile->RecS;                      /* offset in Bytes */
    regs.x.cx = RecNo >> 16;                          /* hi Word Offset */
    regs.x.dx = RecNo & 0xFFFF;                      /* lo Word Offset */
    intdos( &regs, &regs );
    if ( !regs.x.cflag )
        NetError = NE_OK;                            /* no error */
    else
        NetError = regs.x.ax;                        /* note error number */
}

```