

```

***** N E T F I L E *****
*-----*
* Task       : Implements network supporting file functions. *
*-----*
* Author      : Michael Tischer
* Developed on : 09/07/91
* Last update  : 01/29/92
*****

```

```
unit NetFileP;
```

```
interface
```

```
uses Crt, Dos; { Add CRT and DOS units }
```

```
const {-- Types of file access available -----}
```

```

fm_r   = 0; { Read-only }
fm_w   = 1; { Write-only }
fm_rw  = 2; { Read and write in normal Pascal mode }

```

```
{-- Types of file protection -----}
```

```

sm_comp = $00; { Compatibility mode, no file protection }
sm_rw   = $10; { Read and write prohibited by others }
sm_r    = $20; { Read by others permitted, writing prohibited }
sm_w    = $30; { Reading and writing by others permitted }
sm_no   = $40; { All permitted, protected by record lock }

```

```
{-- Possible errors during procedure calls -----}
```

```

NE_OK      = $00; { No error }
NE_FileNotFound = $02; { Error: File not found }
NE_PathNotFound = $03; { Error: Path not found }
NE_TooManyFiles = $04; { Error: Too many open files }
NE_AccessDenied = $05; { Error: Access to file denied }
NE_InvalidHandle = $06; { Error: Invalid file handle }
NE_AccessCode   = $07; { Error: Illegal access code }
NE_Share        = $20; { Violation of Share rights }
NE_Lock         = $21; { Error while (un)locking a record }
NE_ShareBuffer  = $24; { Share buffer overflow }

```

```
var NetError : integer; { Error number from DOS interrupt }
```

```
function ShareInst : boolean; { Share installed? }
```

```
function NetErrorMsg( Number : word ) : string; { Error message }
```

```

procedure NetReset( FName : string; { Open file }
                   AMode : integer;
                   RecS : word;
                   var DFile );

```

```

procedure NetRewrite( FName : string; { Open new file }
                    AMode : integer;
                    RecS : word;
                    var DFile );

```

```
procedure NetClose( var DFile ); { Close file }
```

```

function NetLock( var DFile; { Lock file range }
                RecNo : longint;
                RngNum : longint ) : boolean;

```

```

function NetUnlock( var DFile; { Unlock file range }
                  RecNo : longint;
                  RngNum : longint ) : boolean;

```

```
function Is_NetOpen( var DFile ) : boolean; { Is file open? }
```

```
function Is_NetWriteOk( var DFile ) : boolean; { Writing to file O.K. }
```

```
function Is_NetReadOk( var DFile ) : boolean; { Reading from file O.K. }
```

```

{-- The Read, Write and Seek procedures only work with files set in --}
{-- input-output mode. The following procedures must be used if --}
{-- files must be opened in other modes. --}

```

```

procedure NetWrite( var DFile; { Write to a file }
                  var FData );

```

```

procedure NetRead( var DFile; { Read from a file }
                  var FData );

```

```
procedure NetSeek( var DFile; { Position file pointer }
```

implementation

```

const {-- Function numbers for DOS calls -----}

FCT_OPEN      = $3D;      { Function: Open file with handle }
FCT_CLOSE     = $3E;      { Function: Close file with handle }
FCT_CREATE    = $3C;      { Function: Create file with handle }
FCT_WRITE     = $40;      { Function: Write to file }
FCT_READ      = $3F;      { Function: Read from file }
FCT_LSEEK     = $42;      { Function: Set file pointer }
FCT_REC_LOCK  = $5C;      { Function: Record locking }

{-- Function & interrupt numbers for other interrupt calls -----}

MULTIPLEX     = $2F;      { Multiplex interrupt }
FCT_SHARE     = $1000;    { Install text for Share }

{-- Turbo Pascal file identifiers -----}

fmClosed      = $D7B0;    { File closed }
fmInput       = $D7B1;    { File opened for reading }
fmOutput      = $D7B2;    { File opened for writing }
fmInOut       = $D7B3;    { File opened for reading and writing }

{*****}
{ * ShareInst : Installs test for Share. }
{ * Input : None }
{ * Output : TRUE if Share is installed }
{ * Global var. : NetError/W (error status after call) }
{*****}

function ShareInst : boolean;

var regs : registers; { Processor registers for interrupt call }

begin
  regs.ax := FCT_SHARE; { Test for installed Share }
  intr( MULTIPLEX, regs ); { Call multiplex interrupt }
  ShareInst := ( regs.al = $FF ); { Return result }
  NetError := NE_OK; { No error }
end;

{*****}
{ * NetErrorMsg : Error message text. }
{ * Input : Error number }
{ * Output : Meaning }
{*****}

function NetErrorMsg( Number : word ) : string;

var Sdummy : string;

begin
  case Number of
    NE_OK : NetErrorMsg := 'No error';
    NE_FileNotFound : NetErrorMsg := 'File not found';
    NE_PathNotFound : NetErrorMsg := 'Path not found';
    NE_TooManyFiles : NetErrorMsg := 'Too many files open';
    NE_AccessDenied : NetErrorMsg := 'File access denied';
    NE_InvalidHandle : NetErrorMsg := 'Invalid file handle';
    NE_AccessCode : NetErrorMsg := 'Illegal access code';
    NE_Share : NetErrorMsg := 'Violation of Share rights';
    NE_Lock : NetErrorMsg := 'Error during record lock';
    NE_ShareBuffer : NetErrorMsg := 'Share buffer overflow';
    else
      begin
        str( Number, Sdummy );
        NetErrorMsg := 'DOS error: ' + Sdummy;
      end;
  end;
end;

{*****}
{ * NetCreate : Creates a file. }
{ * Input : Filename, opening mode, record size }
{ * Output : Opened file }
{ * Global var. : NetError/W (error status after call) }
{*****}

procedure NetRewrite( FName : string;
                     AMode : integer;
                     RecS : word;
                     var DFile );

var regs : registers; { Processor registers for interrupt call }

```

```

FName2 : string;                                { Filename for local access }

begin
  FName2 := FName + #0;                          { Copy and prepare filename }
  with regs do
  begin
    ds := seg( FName2[ 1 ] );                    { Assign filename }
    dx := ofs( FName2[ 1 ] );
    ah := FCT_CREATE;                            { Function number: Open file }
    cx := 0;                                     { File attribute }
    msdos( regs );                               { Interrupt call }
    if ( ( flags and fcarry ) = 0 ) then         { Open successful? }
    begin
      bx := ax;                                  { Handle in register BX }
      ah := FCT_CLOSE;                          { Function number: Close file }
      msdos( regs );
      if ( ( flags and fcarry ) = 0 ) then       { Close successful? }
      NetReset( FName, AMode, Recs, DFile )     { Open file }
      else
      NetError := ax;                            { Note error number }
    end
    else
      NetError := ax;                            { Note error number }
    end;
  end;
end;

```

```

{*****}
{ * NetReset      : Opens a specific file.                * }
{ * Input        : Filename, open mode, record size      * }
{ * Output       : Opened file                          * }
{ * Global var.  : NetError/W (error status after call)   * }
{*****}

```

```

procedure NetReset(      FName   : string;
                      AMode   : integer;
                      RecS    : word;
                      var DFile );

```

```

var regs : registers;      { Processor registers for interrupt call }

```

```

begin
  FName := FName + #0;                          { Filename must end with #0 }
  with regs do
  begin
    ds := seg( FName[ 1 ] );                    { Assign filename }
    dx := ofs( FName[ 1 ] );
    ah := FCT_OPEN;                             { Function number: Open file }
    al := AMode;                                { Status byte for access mode and locking }
    msdos( regs );                               { Interrupt call }
    if ( ( flags and fcarry ) = 0 ) then         { Open successful? }
    begin
      NetError := NE_OK;                        { No error }
      with filerec( DFile ) do
      begin
        move( FName[ 1 ], filerec( DFile ).Name, { Assign }
              length( FName ) );                { filename }
        Handle := ax;                           { File handle }
        RecSize := RecS;                        { Specify record size }
        case ( AMode and $0F ) of               { Specify Pascal file mode }
          fm_r   : Mode := fmInput;
          fm_w   : Mode := fmOutput;
          fm_rw  : Mode := fmInOut;
        end;
      end;
    end
    else
    begin
      NetError := ax;                            { Note error number }
      filerec( DFile ).Mode := fmClosed;         { File not opened }
    end;
  end;
end;
end;

```

```

{*****}
{ * NetClose     : Closes a file.                        * }
{ * Input        : File handle                          * }
{ * Output       : None                                  * }
{*****}

```

```

procedure NetClose( var DFile );

```

```

var regs : registers;      { Processor registers for interrupt call }

```

```

begin
  if ( Filerec( DFile ).Mode <> fmClosed ) then { File open? }
  begin

```

```

with regs do
begin
    ah := FCT_CLOSE;                { Function number: Close file }
    bx := FileRec( DFile ).Handle;  { File handle }
    msdos( regs );                  { Interrupt call }
end;
FileRec( DFile ).Mode := fmClosed; { Close file }
NetError := NE_OK;                 { No error }
end
else
    NetError := NE_InvalidHandle;   { File not open }
end;

{*****}
{ * Locking      : Locks or unlocks a file range.                * }
{ * Input       : File handle, operation, offset for start of file, * }
{ *              length of range to be (un)locked                * }
{ * Output      : TRUE if successful                              * }
{ * Global var. : NetError/W (error status after call)           * }
{ * Info        : Call NetLock and NetUnlock for internal access only.* }
{*****}

function Locking( Handle      : word;
                  Operation   : byte;
                  Offset      : longint;
                  WrLen       : longint ) : boolean;

var regs : registers;           { Processor registers for interrupt call }

begin
with regs do
begin
    ah := FCT_REC_LOCK;          { Function number for interrupt call }
    al := Operation;             { 0 = Lock, 1 = Unlock }
    bx := Handle;                { File handle }
    cx := offset shr 16;         { High word offset }
    dx := offset and $FFFF;      { Low word offset }
    si := WrLen shr 16;          { High word length }
    di := WrLen and $FFFF;       { Low word length }
    msdos( regs );               { Interrupt call }
    if ( ( flags and fcarry ) = 0 ) then { Locking successful? }
    begin
        Locking := true;          { No error }
        NetError := NE_OK;
    end
    else
    begin
        Locking := false;
        NetError := ax;           { Note error number }
    end;
end;
end;

{*****}
{ * NetLock      : Locks records.                                  * }
{ * Input       : File, record number, number of records to be locked * }
{ * Output      : TRUE if successful                              * }
{ * Global var. : NetError/W (error status after call)           * }
{*****}

function NetLock( var DFile;
                 RecNo : longint;
                 RngNum : longint ) : boolean;

begin
    NetLock := Locking( filerec( DFile ).Handle, 0,
                       filerec( DFile ).Recsize * RecNo,
                       filerec( DFile ).Recsize * RngNum );
end;

{*****}
{ * NetUnLock    : Unlocks locked records.                        * }
{ * Input       : File, record number, number of records to be locked * }
{ * Output      : TRUE if successful                              * }
{ * Global var. : NetError/W (error status after call)           * }
{*****}

function NetUnlock( var DFile;
                  RecNo : longint;
                  RngNum : longint ) : boolean;

begin
    NetUnLock := Locking( filerec( DFile ).Handle, 1,
                       filerec( DFile ).Recsize * RecNo,
                       filerec( DFile ).Recsize * RngNum );
end;

```

```

*****
{ Is_NetWriteOk : Enables file output. }
{ * Input      : File }
{ * Output     : TRUE if output is enabled }
*****

function Is_NetWriteOk( var DFile ) : boolean;

begin
  with Filerec( DFile ) do
    Is_NetWriteOk := ( Mode = fmOutput ) or ( Mode = fmInOut );
  end;
end;

*****
{ Is_NetReadOk : Enables file input. }
{ * Input      : File }
{ * Output     : TRUE if output is enabled }
*****

function Is_NetReadOk( var DFile ) : boolean;

begin
  with Filerec( DFile ) do
    Is_NetReadOk := ( Mode = fmInput ) or ( Mode = fmInOut );
  end;
end;

*****
{ Is_NetOpen   : Opens file. }
{ * Input      : File }
{ * Output     : TRUE if file is open }
*****

function Is_NetOpen( var DFile ) : boolean;

begin
  with Filerec( DFile ) do
    Is_Netopen := ( Mode = fmInput ) or ( Mode = fmOutput ) or
      ( Mode = fmInOut );
  end;
end;

*****
{ NetWrite     : Writes to file. }
{ * Input      : File, data }
{ * Output     : None }
{ * Info       : Output is only possible in Pascal procedures when }
{ *             files have been opened in input-output mode. }
{ *             Attention: No type checking performed here. }
*****

procedure NetWrite( var DFile;
                   var FData );

var regs : registers;      { Processor registers for interrupt call }

begin
  with regs do
    begin
      ds := seg( FData );          { Data address }
      dx := ofs( FData );
      ah := FCT_WRITE;             { Function number: Write file }
      bx := filerec( DFile ).Handle; { File handle }
      cx := filerec( DFile ).Recsize; { Number of bytes }
      msdos( regs );               { Interrupt call }
      if ( ( flags and fcarry ) = 0 ) then { Write successful? }
        NetError := NE_OK;         { No error }
      else
        NetError := ax;            { Note error number }
      end;
    end;
  end;
end;

*****
{ NetRead      : Reads from file. }
{ * Input      : File, variable for accessing data }
{ * Output     : Data }
{ * Info       : Output is only possible in Pascal procedures when }
{ *             files have been opened in input-output mode. }
{ *             Attention: No type checking performed here. }
*****

procedure NetRead( var DFile;
                  var FData );

var regs : registers;      { Processor registers for interrupt call }

begin
  with regs do

```

```

begin
  ds := seg( FData );
  dx := ofs( FData );
  ah := FCT_READ;
  bx := filerec( DFile ).Handle;
  cx := filerec( DFile ).Recsize;
  msdos( regs );
  if ( ( flags and fcarry ) = 0 ) then
    NetError := NE_OK
  else
    NetError := ax;
  end;
end;

{*****}
{ * NetSeek : Sets file pointer. * }
{ * Input : File, record number * }
{ * Output : None * }
{ * Info : Output is only possible in Pascal procedures when * }
{ * files have been opened in input-output mode. * }
{*****}

procedure NetSeek( var DFile;
                  RecNo : longint );

var regs : registers;

begin
  with regs do
    begin
      ah := FCT_LSEEK;
      al := 0;
      bx := filerec( DFile ).Handle;
      RecNo := RecNo * filerec( DFile ).Recsize;
      cx := RecNo shr 16;
      dx := recNo and $FFFF;
      msdos( regs );
      if ( ( flags and fcarry ) = 0 ) then
        NetError := NE_OK
      else
        NetError := ax;
      end;
    end;
  end;
begin
end.

```