Lesen von NOFAT – SD - Karten am PC

Der Vorteil des NOFAT- Systems ist, dass der Mikrokontroller entlastet wird, da er die ganze Verwaltung der Dateien nicht zu übernehmen braucht. Sowieso werden die Daten ja normalerweise linear hintereinander geschrieben, so dass die komplizierte Clusterverwaltung eigentlich unnütz ist.

Der Nachteil ist, dass man beim Lesen der Karte mit dem PC vor einem Problem steht. Die Karte ist in einem Nicht-DOS-System formatiert, der Inhalt ist für das Betriebssystem zunächst unsichtbar. Hier ist nicht dateimässiges, sondern sektormässiges Lesen angesagt.

Ehrlich gesagt hatte ich mir die Lösung dieser Aufgabe einfacher vorgestellt. Schliesslich gibt es BIOS-Interrupts, die genau dies erledigen, und die Windows-API müsste ja auch über die entsprechenden Routinen verfügen, sonst könnte sie z.B. kein Laufwerk formatieren.

Um es kurz zu machen: der Weg war lang und dornig. Nicht etwa weil die Sache kompliziert an sich ist, sondern weil die benötigten Informationen spärlich und schwierig zu finden sind (und da die BIOS-Interrupts in VB nicht (?) nutzbar sind.

Hier ein möglicher Weg in Visual Basic 5, der nur 3 API-Funktionen benutzt.

Das Lesen geschieht in 3 Schritten für die 3 Unterprogramme geschrieben wurden:
Ein ganz einfaches Programm könnte so aussehen:

```
'1. Dateihandle öffnen
   GetDriveHandle Drivename$

'2. Bytes lesen
   ReadSectorFromDrive buffer()                    'liest Sektor 0

   'und anzeigen
   DisplaySector buffer()

   '.........diese Schrite wiederholen um weitere Sektoren zu lesen und anzuzeigen


'3. Handle freigeben
  CloseDriveHandle
```

Es wird zum Aufnehmen der Sektorbytes ein Array buffer(512) as byte benutzt.

Hier nun die einzelnen SUBs, inklusive einer rudimentären Fehlerbehandlung:

```
Sub GetDriveHandle(Drivename$)
'muss zuerst aufgerufen werden um Zugang zum Drive zu bekommen

'Fehlervariablen zurücksetzen
CreateFileError = 0
ReadFileError = 0

filename$ = "\\.\" & Drivename$
lngFileHandle = CreateFile(filename$, GENERIC_READ Or GENERIC_WRITE,
FILE_SHARE_READ, ByVal 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, ByVal 0)

   'Bei Fehlschlag aussteigen
```

```
   If (lngFileHandle = INVALID_HANDLE_VALUE) Then
      MsgBox "Invalid handle!"
      CreateFileError = 1

      Exit Sub
   Else
      'MsgBox "File handle:" & Str$(lngFileHandle)
   End If

End Sub
```

Es wird CreateFile benutzt, um Zugang zum Laufwerk zu bekommen.
Aus der Bezeichnung ist nicht ohne weiteres zu sehen, dass es um sektorweises Lesen geht. Man hätte eher eine API-Funktion wie „Readsector“ o.ä. erwartet, aber die gibt es nicht.

Wichtig ist das „\\.\“ vor dem Laufwerksnamen, für das ich keine Dokumentation gefunden habe, aber genau diese Schreibweise erlaubt den sektorweisen Zugriff.

Achtung:
Laufwerke die Windows in Beschlag genommen hat (wie C:\) können kein Handle bekommen und damit mit dieser Methode nicht gelesen werden.
Deswegen sollten auch etwaige Explorer-Fenster, die sich auf das Karten-Laufwerk beziehen, zuerst geschlossen werden.

Ist ein Handle da, können Sektoren gelesen werde, und zwar inkremental, d.h. bei Sektor 0 beginnend:

```
Sub ReadSectorFromDrive(buffer() As Byte)
'liest 512 Bytes (1 Sektor) in buffer() ein
   If CreateFileError Then
      ReadFileError = 1
      Exit Sub
   End If

   retReadFile& = ReadFile(lngFileHandle, buffer(0), 512, Bytesread&, 0&)
   If retReadFile& = 0 Then
      MsgBox "Readfile returns " & Str$(retReadFile&) & vbCrLf & "Read Error!"
      ReadFileError = 1
      Exit Sub
   End If
End Sub
```

Jeder weitere Aufruf des SUBs liest einen neuen Sektor ein.

Zum Schluss darf man nicht vergessen, das Handle wieder freizugeben, sonst ist das Laufwerk danach nicht mehr ansprechbar:

```
Sub CloseDriveHandle()
          CloseHandle lngFileHandle
End Sub
```

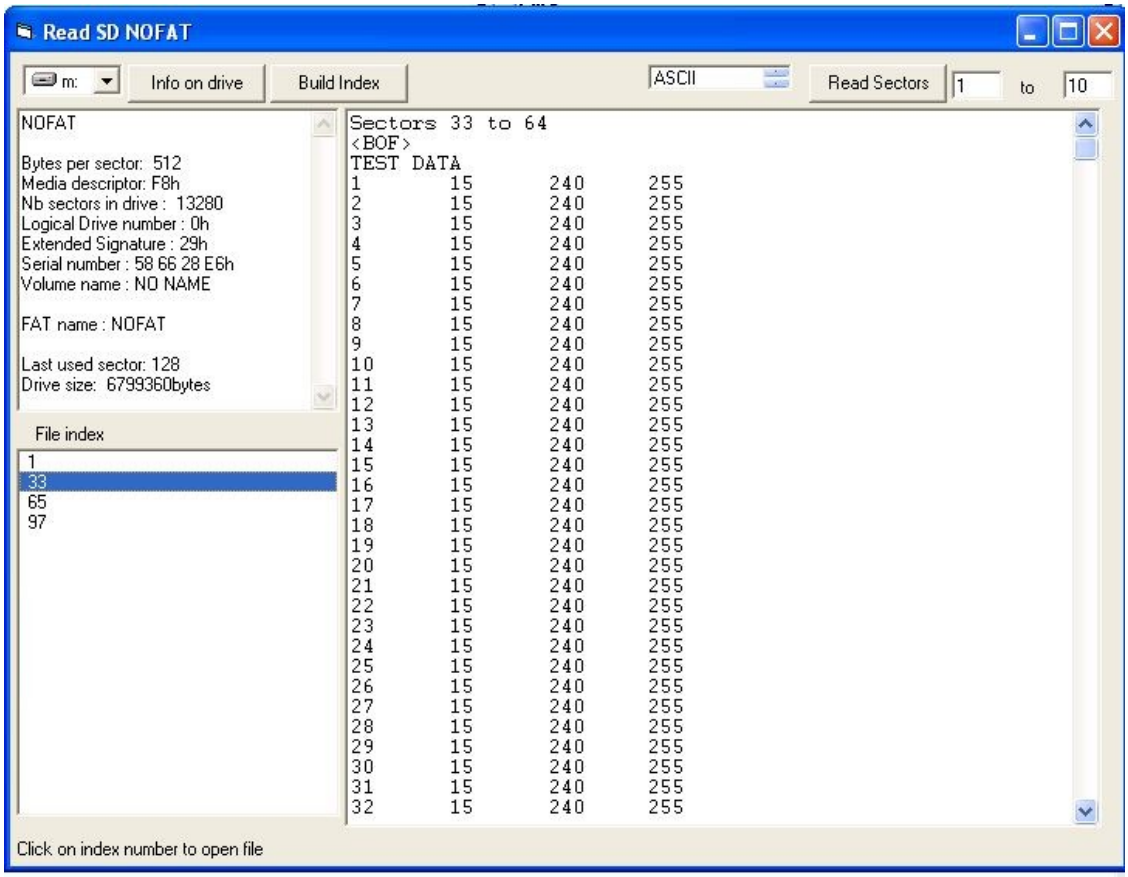
Damit das Ganze funktioniert, benötigt man natürlich einige globale Variablen für die SUBs und die API-Deklarationen, sowie die zugehörigen API-Konstanten:

```
'Windows-Variablen / Konstanten:
Private Type OVERLAPPED
   Internal             As Long
   InternalHigh         As Long
   offset               As Long
   OffsetHigh           As Long
   hEvent               As Long
End Type

Private Const GENERIC_READ = &H80000000
Private Const GENERIC_WRITE = &H40000000
```

```
Private Const OPEN_EXISTING = 3
Private Const INVALID_HANDLE_VALUE = -1
Private Const FILE_DEVICE_FILE_SYSTEM = &H9
Private Const METHOD_BUFFERED = 0
Private Const FILE_READ_DATA = (&H1)
Private Const FILE_WRITE_DATA = (&H2)

'API-Deklarationen
Private Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" _
   (ByVal lpFileName As String, ByVal dwDesiredAccess As Long, _
   ByVal dwShareMode As Long, lpSecurityAttributes As Any, _
   ByVal dwCreationDisposition As Long, ByVal dwFlagsAndAttributes _
   As Long, _
   ByVal hTemplateFile As Long) As Long

Declare Function ReadFile Lib "kernel32" (ByVal hFile As Long, _
       lpBuffer As Any, ByVal nNumberOfBytesToRead As Long, _
       lpNumberOfBytesRead As Long, ByVal lpOverlapped As Long) As Long

Private Declare Function CloseHandle Lib "kernel32" _
   (ByVal hObject As Long) As Long

'Im Modul benötigt:
Dim lngFileHandle As Long      'Handle für Drive
Dim CreateFileError            'Fehler beim Erzeugen des Handles
Dim ReadFileError              'Fehler beim Lesen vom Laufwerk
```

Das Ganze kann natürlich jetzt weiter verfeinert werden, so dass man Informationen über das Laufwerk oder eben NOFAT - „Dateien“ lesen kann.

Hier ein Screenshot einer provisorischen Version des NOFAT Readers:



Links oben Infos über das Laufwerk (die SD-Karte), links unten der File Index, in dem die Startsektoren der „Dateien“ stehen. Durch Doppelklick auf einen Index wird der „Datei“-Inhalt angezeigt (rechts)