


[Home Page](#) 
[Informazioni](#) 
[Aiuto](#) 

Ottenere informazioni su un'unità disco FAT12/16

http://www.vbsimple.net/howto/ht_050.htm

Difficoltà:  4 / 5

Uno dei più pesanti limiti di Visual Basic è quello di non permettere un accesso diretto alle unità disco ad esempio per recuperare lo spazio libero in un disco floppy. Per queste finalità ci viene incontro l'[API](#) ma la sua gestione è abbastanza complessa.

Questo articolo si focalizza sul recupero delle informazioni generali su un'unità disco FAT12 o FAT16, quali i dischi floppy o le vecchie partizioni MS-DOS. **Non funziona correttamente** con unità FAT32, NTFS o CD; per questi sistemi sarà utilizzata una seconda soluzione, leggermente più lenta di quella presentata in quest'articolo.

Il codice non è molto semplice pertanto sarà spezzato in due forms ed un modulo di classe; cominceremo proprio da quest'ultimo:

```
1. Option Explicit
2.
3. Private m_VolumeLabel As String
4. Private m_SerialNumber As Long
5. Private m_SectorsPerCluster As Long
6. Private m_BytesPerSector As Long
7. Private m_TotalClusters As Long
8. Private m_FreeClusters As Long
9. Private m_FileSystem As String
10. Private m_SelectedDrive As Long
11.
```

I membri privati dichiarati alle righe 3-10 saranno restituiti dalle proprietà pubbliche della [classe](#) e non saranno inizialmente affrontate.

```
12. Private Declare Function GetDiskFreeSpace Lib "kernel32" Alias
    "GetDiskFreeSpaceA" (ByVal lpRootPathName As String, lpSectorsPerCluster As Long,
    lpBytesPerSector As Long, lpNumberOfFreeClusters As Long, lpTotalNumberOfClusters
    As Long) As Long
13. Private Declare Function GetVolumeInformation Lib "kernel32" Alias
    "GetVolumeInformationA" (ByVal lpRootPathName As String, ByVal lpVolumeNameBuffer
    As String, ByVal nVolumeNameSize As Long, lpVolumeSerialNumber As Long,
    lpMaximumComponentLength As Long, lpFileSystemFlags As Long, ByVal
    lpFileSystemNameBuffer As String, ByVal nFileSystemNameSize As Long) As Long
```

La funzione *GetDiskFreeSpace*, nonostante il nome, non restituisce lo spazio disponibile del disco ma riporta una serie di informazioni utili quali il numero di settori totali, il numero di [bytes](#) per settore, il numero di cluster totali e liberi. I suddetti dati saranno naturalmente memorizzati nelle variabili membro viste in precedenza.

La funzione *GetVolumeInformation* recupera altre informazioni non restituite dalla funzione precedente, quale l'etichetta, il numero di serie ed il tipo di [file system](#) del volume.

```

13. Private Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" (ByVal
    lpFileName As String, ByVal dwDesiredAccess As Long, ByVal dwShareMode As Long,
    ByVal lpSecurityAttributes As Long, ByVal dwCreationDisposition As Long, ByVal
    dwFlagsAndAttributes As Long, ByVal hTemplateFile As Long) As Long
14. Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long
15. Private Declare Function DeviceIoControl Lib "kernel32" (ByVal hDevice As Long,
    ByVal dwIoControlCode As Long, ByRef lpInBuffer As Any, ByVal nInBufferSize As
    Long, ByRef lpOutBuffer As Any, ByVal nOutBufferSize As Long, ByRef lpBytesReturned
    As Long, ByVal lpOverlapped As Long) As Long
16.

```

La funzione *CreateFile* dichiarata alla riga 13 è in realtà una delle funzioni più complesse del sistema operativo e, nonostante il nome, non serve unicamente a creare un nuovo file, ma, fra l'altro, è in grado di restituire un [handle](#) ad un file oppure ad un driver di periferica virtuale per svolgere funzioni di accesso diretto all'hardware. Necessita non solo del *'file'* da aprire ma richiede anche una serie di valori di accesso, per richiedere ad esempio la lettura o la scrittura.

La funzione *CloseHandle*, trattata in [un altro HowTo](#), molto semplicemente chiude un handle aperto da un'altra funzione quale la *CreateFile*.

L'ultima funzione, *DeviceIoControl*, consente di effettuare operazioni di input ed output sopra un handle; in questo esempio servirà per leggere dati da un settore dell'unità disco specificando la costante *VWIN32_DIOC_DOS_INT25* dichiarata successivamente.

```

17. Private Const GENERIC_READ = &H80000000
18. Private Const FILE_SHARE_READ = &H1&
19. Private Const FILE_SHARE_WRITE = &H2&
20. Private Const OPEN_EXISTING = &H3&
21. Private Const INVALID_HANDLE_VALUE = -1&
22. Private Const VWIN32_DIOC_DOS_INT25 = 2&
23.

```

Le [costanti](#)  API dichiarate alle righe 17-22 saranno utilizzate dalle precedenti funzioni API. Le prime 5 costanti saranno utilizzate dalla funzione *CreateFile*, mentre la costante *VWIN32_DIOC_DOS_INT25* sarà utilizzata dalla funzione *DeviceIoControl* per leggere uno o più settori dall'unità disco.

```

24. Private Type DIOC_REGISTER
25.     reg_EBX As Long
26.     reg_EDX As Long
27.     reg_ECX As Long
28.     reg_EAX As Long
29.     reg_EDI As Long
30.     reg_ESI As Long
31.     reg_Flags As Long
32. End Type
33.

```

La struttura *DIOC_REGISTER* simula il gruppo di [registri di un processore Intel x86](#) e verrà utilizzata dalla funzione API *DeviceIoControl* per inviare i dati in input sulla richiesta e riceverne in uscita i valori di ritorno.

```

34. Public Property Let Drive(ByVal newDrive As String)
35.     m_SelectedDrive = Asc(UCase(newDrive)) - Asc("A")
36. End Property
37.
38. Public Property Get Drive() As String
39.     Drive = Chr$(m_SelectedDrive + Asc("A")) & ":"
40. End Property

```

41.

Di tutte le proprietà della classe, **Drive** è l'unica con possibilità di modifica: consente infatti di cambiare l'unità disco da cui estrarre le informazioni. Poiché la funzione *DeviceIoControl* utilizza un valore numerico per l'unità disco è necessario che il percorso dell'unità fornito alla proprietà venga trasformato in indice numerico. Così l'unità A: diventa 0, l'unità C: diventa 2...

```

42. Public Property Get BytesPerCluster() As Long
43.     BytesPerCluster = m_BytesPerSector * m_SectorsPerCluster
44. End Property
45.
46. Public Property Get BytesPerSector() As Long
47.     BytesPerSector = m_BytesPerSector
48. End Property
49.
50. Public Property Get Clusters() As Long
51.     Clusters = m_TotalClusters
52. End Property
53.
54. Public Property Get DiskFreeSize() As Long
55.     DiskFreeSize = m_FreeClusters * m_SectorsPerCluster * m_BytesPerSector
56. End Property
57.
58. Public Property Get DiskOccupiedSize() As Long
59.     DiskOccupiedSize = (m_TotalClusters - m_FreeClusters) * m_SectorsPerCluster *
    m_BytesPerSector
60. End Property
61.
62. Public Property Get DiskSize() As Long
63.     DiskSize = m_TotalClusters * m_SectorsPerCluster * m_BytesPerSector
64. End Property
65.
66. Public Property Get FileSystem() As String
67.     FileSystem = m_FileSystem
68. End Property
69.
70. Public Property Get FreeClusters() As Long
71.     FreeClusters = m_FreeClusters
72. End Property
73.
74. Public Property Get Sectors() As Long
75.     Sectors = m_TotalClusters * m_SectorsPerCluster
76. End Property
77.
78. Public Property Get SectorsPerCluster() As Long
79.     SectorsPerCluster = m_SectorsPerCluster
80. End Property
81.
82. Public Property Get SerialNumber() As String
83.     SerialNumber = Right$(String$(8, Asc("0")) & Hex(m_SerialNumber), 8)
84.     SerialNumber = Left$(SerialNumber, 4) & "-" & Mid$(SerialNumber, 5)
85. End Property
86.
87. Public Property Get VolumeName() As String
88.     VolumeName = m_VolumeLabel
89. End Property
90.

```

Tutte le altre proprietà, in sola lettura, non fanno altro che restituire i dati membro della classe, in maniera semplice o effettuando delle banali operazioni di conversione, come ad esempio la trasformazione del numero di serie (proprietà **SerialNumber**) da valore numerico decimale a valore esadecimale. Le altre conversioni sono semplici moltiplicazioni quale il numero di cluster totali per il numero di settori per cluster, al fine di ottenere il numero totale di settori dell'unità disco.

```

91. Public Sub TakeSnapshot(Optional ByVal strDrive As String = "")
92.     Dim lngSysFlags As Long
93.     If Len(strDrive) = 0 Then strDrive = Chr$(m_SelectedDrive + Asc("A"))
94.     m_FileSystem = String$(255, 0)
95.     m_VolumeLabel = String$(255, 0)
96.     m_BytesPerSector = 0
97.     m_FreeClusters = 0
98.     m_SectorsPerCluster = 0
99.     m_SerialNumber = 0
100.    m_TotalClusters = 0
101.    Call GetDiskFreeSpace(strDrive & ":\", m_SectorsPerCluster, m_BytesPerSector,
m_FreeClusters, m_TotalClusters)
102.    Call GetVolumeInformation(strDrive & ":\", m_VolumeLabel, 255, m_SerialNumber,
ByVal 0&, lngSysFlags, m_FileSystem, 255)
103.    m_VolumeLabel = Left$(m_VolumeLabel, InStr(1, m_VolumeLabel, Chr$(0)) - 1)
104.    m_FileSystem = Left$(m_FileSystem, InStr(1, m_FileSystem, Chr$(0)) - 1)
105. End Sub
106.

```

Il metodo  **TakeSnapshot** di fatto effettua una fotografia della situazione dell'unità disco, recuperando i valori non variabili quali il numero di settori o di cluster e quelli variabili quale lo spazio libero ed occupato dell'unità scelta. Il metodo deve essere richiamato prima della lettura delle proprietà della classe poiché i loro valori dipendono dall'ultima istantanea scattata e non sono aggiornati automaticamente con il cambiamento dei dati dell'unità.

Le righe 94-100 reinizializzano i valori membro della classe, mentre le righe 101 e 102 mediante *GetDiskFreeSpace* e *GetVolumeInformation*, recuperano nuovamente questi valori dall'unità disco scelta.

```

107. Public Function ReadSector(ByVal lngSector As Long) As Variant
108.     Dim hDrive As Long
109.     Dim nRead As Long
110.     Dim bytBuffer() As Byte
111.     Dim inReg As DIOC_REGISTER
112.     Dim outReg As DIOC_REGISTER
113.     hDrive = CreateFile("\\.\VWIN32", GENERIC_READ, FILE_SHARE_READ Or
FILE_SHARE_WRITE, 0&, OPEN_EXISTING, 0&, 0&)
114.     If hDrive = INVALID_HANDLE_VALUE Then
115.         MsgBox "Errore nell'apertura del vwin32 vxd"
116.     Else
117.         TakeSnapshot
118.         If m_BytesPerSector > 0 Then ReDim bytBuffer(m_BytesPerSector - 1) As Byte
119.         inReg.reg_EAX = m_SelectedDrive
120.         inReg.reg_ECX = 1
121.         inReg.reg_EBX = VarPtr(bytBuffer(0))
122.         inReg.reg_EDX = lngSector
123.         Call DeviceIoControl(hDrive, VWIN32_DIOC_DOS_INT25, inReg, Len(inReg),
outReg, Len(outReg), nRead, 0&)
124.         CloseHandle hDrive
125.         ReadSector = bytBuffer
126.     End If
127. End Function

```

L'ultima funzione della classe è anche la più complessa dell'interno modulo: **ReadSector** legge un settore dall'unità disco scelta e per far questo è innanzitutto necessario richiedere un handle al driver di periferica virtuale **VWIN32**, in Windows 9x, responsabile delle comunicazioni a basso livello con le unità disco. Ottenuto tale handle tramite *CreateFile* (riga 113), sarà quindi necessario invocare l'interrupt DOS 25 (riga 123), specificando nel registro **EAX** il drive scelto, nel registro **ECX** il numero di settori da leggere (sempre 1), nel registro **EBX** il [puntatore](#) all'area dati già [allocata](#) per contenere i dati ed infine nel registro **EDX** il settore iniziale da cui leggere i dati.

Ottenuti questi dati non dovremo dimenticarci di chiudere l'handle aperto in precedenza (riga 124).

La funzione restituisce un oggetto Variant per permettere anche agli utenti di VB5 di usufruire di questo codice, poiché la versione 5 di Visual Basic non supporta la restituzione di matrici.

La dimostrazione d'uso della classe è molto semplice: un unico form conterrà un controllo *DriveListBox* per la scelta dell'unità disco da cui recuperare le informazioni ed una serie di *Label* poste all'interno di una matrice di controlli, che andranno a mostrare i dati recuperati dalla classe.

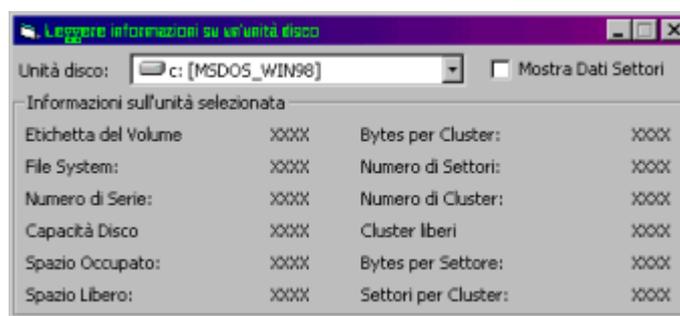
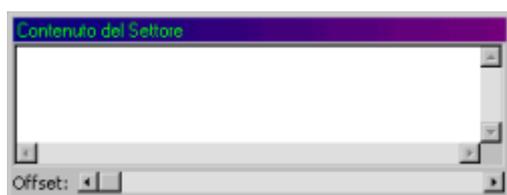


Figura 1

Alla destra della casella di scelta dell'unità è presente anche una *CheckBox* dalla funzione un po' particolare che vedremo in seguito.

Non sarà trattato il codice del form riga per riga perché richiederebbe troppo spazio, del tutto superfluo, vista la semplicità generale. In linea generale all'avvio del form viene [istanziata](#) una variabile della classe *clsFBIDiskInfoFat16* appena sviluppata ed in occasione del cambio dell'unità disco nella *DriveListBox*, viene richiamato il metodo **TakeSnapshot** dell'istanza e sono mostrate una per una le sue proprietà mediante le varie *Label*.

All'avvio del form principale viene anche caricato un secondo Form: *frmDumpedData*, utilizzato per mostrare i dati binari del settore del disco scelto. La sua utilità è mostrata nell'evento  Click del controllo **chkShowSector**; infatti in base al suo valore sarà mostrato o nascosto il secondo form con i dati estratti. Nel momento in cui il form è mostrato viene richiamato anche un metodo pubblico di nome **RefreshData** del form stesso.

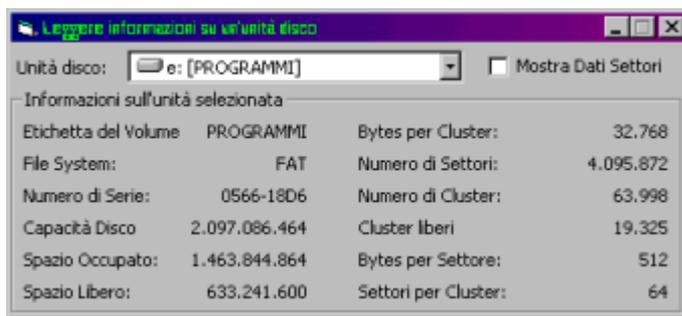


Il form *frmDumpedData* si compone di una semplice *TextBox* multiriga e di una barra di scorrimento orizzontale al cui cambio del valore corrisponde la lettura di un settore dell'unità disco scelta mediante il richiamo al già accennato metodo **RefreshData**.

Il metodo **RefreshData** di fatto legge un settore dell'unità disco mediante il metodo **ReadSector** dell'istanza *DiskInformation* dichiarata nel primo form. Tali dati saranno quindi in seguito formattati in maniera da rappresentarli all'interno della *TextBox*, sia in

esadecimale che sotto forma di codici binari.

Il funzionamento del progetto è molto semplice: lanciato il form principale basterà scegliere un'unità disco con [file system](#) FAT12 o FAT16 (le uniche realmente supportate da questo codice) per recuperarne delle informazioni affidabili.



Ad esempio nella figura a fianco è stata scelta l'unità disco E: della capacità di 2 GB, con circa 630 MB di spazio libero. Sono naturalmente mostrate anche tutte le altre informazioni sulla struttura della partizione. È importante tenere sott'occhio il valore specificato come *File System* poiché tali valori possono essere falsati con unità disco FAT32, NTFS o altro formato.

Selezionando la casella di controllo in alto sarà mostrato il contenuto di un settore per volta dell'unità disco scelta.

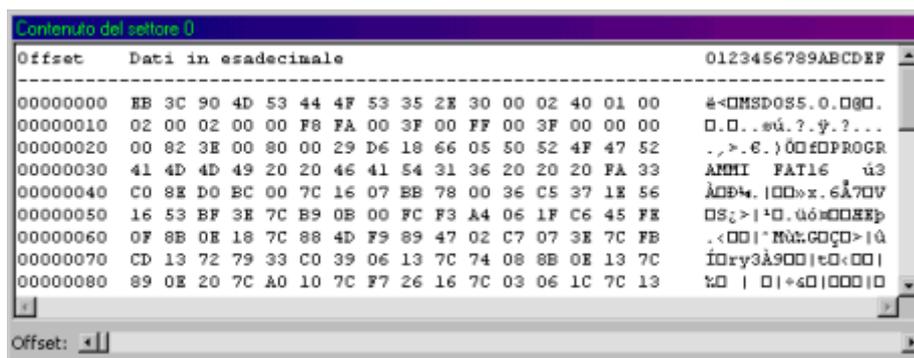


Figura 4

La Figura 4 ad esempio mostra il settore 0, ovvero il primo, dell'unità disco scelta in precedenza. Mediante la barra di scorrimento in basso è naturalmente possibile scegliere un altro settore.

Purtroppo a causa di un limite intrinseco del controllo *HScrollBar* non è possibile assegnare un valore superiore a 32767 e quindi la lettura è limitata a solo questi valori. Limite che naturalmente può essere superato via codice richiamando il metodo **ReadSector** con il numero del settore da leggere.

Il codice mostrato è sicuramente complesso e soggetto a parecchie trappole visto che il passaggio di un semplice valore sbagliato alla funzione *DeviceIoControl* è da solo sufficiente a generare un errore irreversibile ([BSOD](#)) nel sistema che obbliga a riavviare il computer.

Si raccomanda pertanto di non abusare di queste funzioni, molto delicate, e non apportare modifiche se non strettamente necessario e con piena coscienza di ciò che si sta facendo. Come già detto il passaggio di un solo valore errato può provocare danni lievi quali il

riavvio della macchina oppure la scrittura di dati in aree di sistema fino al completo blocco della macchina.

Un limite molto pesante del suddetto codice è quello di recuperare informazioni esclusivamente da partizioni FAT12 e FAT16. Sono quindi esclusi tutti gli altri File System e le informazioni recuperati intorno a questi altri sistemi possono risultare errate. L'altro grosso limite riguarda il sistema operativo utilizzato per eseguire il codice: questo codice infatti non funziona nei sistemi Windows NT ma soltanto nella famiglia Windows 9x.

[Fibia FBI](#)

6 Novembre 2002



[Torna all'indice degli HowTo](#)
