Utilizzare il registro di Windows (prima parte) API

http://www.vbsimple.net/howto/ht_044.htm

Difficoltà: 5 / 5

Premessa:
Il codice trattato in questo articolo è stato riscritto per renderlo più intuitivo possibile e per utilizzarlo all'interno di un modulo di classe, aumentando e semplificando quindi la sua riusabilità. È stato studiato intorno a *Windows 95/98/ME* e tralascia alcuni aspetti fondamentali necessari per l'utilizzo in *Windows NT/2000*.

Quasi certamente in futuro il codice verrà nuovamente modificato per renderlo compatibile anche con gli altri sistemi ed implementare così tutte quelle caratteristiche non ancora presenti in questa seconda revisione.

Il giudizio di complessità massimo non deve spaventare perché in gran parte dovuto alla lunghezza e completezza del modulo di classe sviluppato ed in parte alla necessaria *poca dettagliatezza* che altrimenti avrebbe portato via molto più spazio del dovuto.

Un problema comunissimo per tantissimi programmi è il salvataggio ed il caricamento delle opzioni del programma sviluppato. Non è molto sensato, infatti, obbligare ogni volta l'utente a regolarsi i parametri del programma secondo le sue necessità; è molto più agevole far sì che all'uscita del programma tali parametri ed impostazioni vengano salvate da qualche parte e riprese al successivo ricaricamento del programma.

Esistono due soluzioni fondamentali per il salvataggio ed il ripristino delle impostazioni: la prima consiste nell'utilizzare i classici ed antiquati files di impostazioni con estensione **INI**, mentre la seconda sfrutta il registro di Windows (chiamato talvolta semplicemente *Registry*), un particolare file di Windows contenente molte impostazioni di quasi tutti i moderni programmi per Windows.

All'interno del registro sono contenute tantissime informazioni e talvolta è necessario accedervi per recuperare tali informazioni in mancanza di una funzione API che li ricavi direttamente. Si raccomanda quindi la manipolazione del registry soltanto come ultima possibilità per il recupero dei dati, preferendo quindi, l'utilizzo delle funzioni API specializzate.

In questo lungo tutorial svilupperemo una classe per la lettura, la modifica, l'aggiunta e l'eliminazione di chiavi e valori del registro. Sono state aggiunte in seguito due funzioni per l'esportazione e l'importazione di parti del registro su file compatibili con Regedit. Si suppone una minima conoscenza della struttura del Registry.
Tralasciamo le spiegazioni e vediamo subito il codice denso di API:


```
Option Explicit
Option Base 0

Private Const STANDARD_RIGHTS_ALL As Long = &H1F0000
Private Const STANDARD_RIGHTS_READ As Long = &H20000
Private Const STANDARD_RIGHTS_WRITE As Long = &H20000
Private Const SYNCHRONIZE As Long = &H100000
Private Const KEY_QUERY_VALUE As Long = &H1
Private Const KEY_SET_VALUE As Long = &H2
Private Const KEY_CREATE_SUB_KEY As Long = &H4
Private Const KEY_ENUMERATE_SUB_KEYS As Long = &H8
Private Const KEY_NOTIFY As Long = &H10
Private Const KEY_CREATE_LINK As Long = &H20

Private Const REG_OPTION_NON_VOLATILE As Long = 0

```

Si apre il sipario con la presentazione di una serie di costanti API utilizzate più avanti: sono suddivise in due tipologie: quelle relative alla sicurezza e quella indicativa del tipo di dati da trattare ovvero **non volatili**, i normali dati del registro opposti a quelli volatili la cui esistenza si spegne al riavvio del computer. Vedremo le altre costanti nelle enumerazioni successive.

```
Private Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias
"RegOpenKeyExA" (ByVal hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As
Long, ByVal samDesired As Long, phkResult As Long) As Long
Private Declare Function RegCreateKeyEx Lib "advapi32.dll" Alias
"RegCreateKeyExA" (ByVal hKey As Long, ByVal lpSubKey As String, ByVal Reserved As
Long, ByVal lpClass As String, ByVal dwOptions As Long, ByVal samDesired As Long,
ByVal lpSecurityAttributes As Long, phkResult As Long, lpdwDisposition As Long) As
Long
Private Declare Function RegCloseKey Lib "advapi32.dll" (ByVal hKey As Long) As
Long
Private Declare Function RegFlushKey Lib "advapi32.dll" (ByVal hKey As Long) As
Long
Private Declare Function RegDeleteKey Lib "advapi32.dll" Alias
"RegDeleteKeyA" (ByVal hKey As Long, ByVal lpSubKey As String) As Long
Private Declare Function RegEnumKeyEx Lib "advapi32.dll" Alias
"RegEnumKeyExA" (ByVal hKey As Long, ByVal dwIndex As Long, ByVal lpName As String,
lpcbName As Long, lpReserved As Long, ByVal lpClass As String, lpcbClass As Long,
ByVal lpftLastWriteTime As Long) As Long
Private Declare Function RegQueryInfoKey Lib "advapi32.dll" Alias
"RegQueryInfoKeyA" (ByVal hKey As Long, ByVal lpClass As String, lpcbClass As Long,
lpReserved As Long, lpcSubKeys As Long, lpcbMaxSubKeyLen As Long, lpcbMaxClassLen
As Long, lpcValues As Long, lpcbMaxValueNameLen As Long, lpcbMaxValueLen As Long,
lpcbSecurityDescriptor As Long, lpftLastWriteTime As Long) As Long
```

Segue un primo elenco di dichiarazioni di funzioni API relative alle chiavi del registro. Non potendoci soffermare su ognuna accenneremo soltanto qualcosa sul loro funzionamento generale:

- **RegOpenKeyEx**
  Effettua l'apertura di una chiave o di una sua sottochiave.

- **RegCreateKeyEx**
  Crea ed apre una nuova sottochiave della chiave indicata.

- **RegCloseKey**
  Chiude una chiave precedentemente aperta liberando l'handle assegnato.

- **RegFlushKey**

Aggiorna i dati di una chiave e forzando il sistema operativo a scrivere tutte le modifiche non ancora scritte sul disco.

- **RegDeleteKey**
  Elimina una chiave e tutte le sue sottochiavi ed i relativi valori (solo su Windows 9x).

- **RegEnumKeyEx**
  Recupera i nomi di una o più sottochiavi della chiave indicata.

- **RegQueryInfoKey**
  Ricava informazioni su una determinata chiave come il numero di sottochiavi o di valori presenti.

```
24. Private Declare Function RegQueryValueEx Lib "advapi32.dll" Alias
    "RegQueryValueExA" (ByVal hKey As Long, ByVal lpValueName As String, ByVal
    lpReserved As Long, lpType As Long, lpData As Any, lpcbData As Long) As Long
25. Private Declare Function RegSetValueEx Lib "advapi32.dll" Alias
    "RegSetValueExA" (ByVal hKey As Long, ByVal lpValueName As String, ByVal Reserved
    As Long, ByVal dwType As Long, lpData As Any, ByVal cbData As Long) As Long
26. Private Declare Function RegEnumValue Lib "advapi32.dll" Alias
    "RegEnumValueA" (ByVal hKey As Long, ByVal dwIndex As Long, ByVal lpValueName As
    String, lpcbValueName As Long, ByVal lpReserved As Long, lpType As Long, lpData As
    Byte, lpcbData As Long) As Long
27. Private Declare Function RegDeleteValue Lib "advapi32.dll" Alias
    "RegDeleteValueA" (ByVal hKey As Long, ByVal lpValueName As String) As Long
28.
```

Dopo le funzioni relative alle chiavi segue l'elenco delle funzioni relative ai valori contenuti nelle chiavi stesse:

- **RegQueryValueEx**
  Interroga un valore di cui si conosce il nome ottenendo in risposta il suo contenuto ed il suo tipo di dati.

- **RegSetValueEx**
  Modifica il contenuto di un valore esistente e crea un nuovo valore nel caso che esso non dovesse esistere, con il tipo di dati specificato.

- **RegEnumValue**
  Interroga una chiave aperta per richiedere il nome di un valore in essa contenuti e contestualmente consente di recuperare anche il loro contenuto ed il tipo di dati.

- **RegDeleteValue**
  Effettua semplicemente l'eliminazione del valore specificato. Se il valore da eliminare è quello predefinito della chiave ne sarà cancellato soltanto il suo contenuto.

Si sottolinea il fatto che tutte le funzioni che utilizzano stringhe si riferiscono alla versione ANSI piuttosto che alla versione Unicode come indicato dalla A finale al nome della funzione dopo la parola chiave "Alias"; la versione Unicode avrebbe il suffisso W.

Pertanto ogni qual volta viene passata una stringa sotto forma di array di bytes è necessario

fare la conversione di formato tramite la funzione *StrConv*.

Seguono le enumerazioni scritte per semplificare l'utilizzo delle varie costanti API e poter pertanto usufruire del sistema Intellisense.

```
29. Public Enum ChiaviPrincipali
30.   CHIAVE_APERTA = 0
31.   HKEY_CLASSES_ROOT = &H80000000
32.   HKEY_CURRENT_CONFIG = &H80000005
33.   HKEY_CURRENT_USER = &H80000001
34.   HKEY_DYN_DATA = &H80000006
35.   HKEY_LOCAL_MACHINE = &H80000002
36.   HKEY_PERF_ROOT = HKEY_LOCAL_MACHINE
37.   HKEY_PERFORMANCE_DATA = &H80000004
38.   HKEY_USERS = &H80000003
39. End Enum
40.
```

L'enumerazione ***ChiaviPrincipali*** verrà utilizzata da molte funzioni della classe per determinare quale chiave utilizzare: quella aperta dalla nostra istanza (riga 30) oppure una delle chiavi di sistema (righe 31-38) sempre aperte. In funzione della struttura della classe il programma potrà anche fornire una chiave arbitraria, ovvero un handle ad una chiave già aperta.

```
41. Public Enum ChiaviSecurity
42.   KEY_UNKNOWN = 0
43.   KEY_READ = ((STANDARD_RIGHTS_READ Or KEY_QUERY_VALUE Or KEY_ENUMERATE_SUB_KEYS Or
    KEY_NOTIFY) And (Not SYNCHRONIZE))
44.   KEY_WRITE = ((STANDARD_RIGHTS_WRITE Or KEY_SET_VALUE Or KEY_CREATE_SUB_KEY) And
    (Not SYNCHRONIZE))
45.   KEY_ALL_ACCESS = ((STANDARD_RIGHTS_ALL Or KEY_QUERY_VALUE Or KEY_SET_VALUE Or
    KEY_CREATE_SUB_KEY Or KEY_ENUMERATE_SUB_KEYS Or KEY_NOTIFY Or KEY_CREATE_LINK) And
    (Not SYNCHRONIZE))
46. End Enum
47.
```

L'enumerazione ***ChiaviSecurity*** trova il suo vero significato in Windows NT/2000 ed indica 4 permessi standard: da sconosciuto ad accesso massimo. In questa versione della classe studiata appositamente per Windows 95/98/ME ha poco senso di esistere e può tranquillamente essere ignorata.

```
48. Public Enum TipoValoriRegistro
49.   REG_NONE = 0
50.   REG_SZ = 1
51.   REG_EXPAND_SZ = 2
52.   REG_BINARY = 3
53.   REG_DWORD = 4
54.   REG_DWORD_LITTLE_ENDIAN = 4
55.   REG_DWORD_BIG_ENDIAN = 5
56.   REG_LINK = 6
57.   REG_MULTI_SZ = 7
58.   REG_RESOURCE_LIST = 8
59. End Enum
60.
```

L'ultima enumerazione è ***TipoValoriRegistro*** e rappresenta tutti i possibili tipi di dati presenti nel registro. Nel 90% dei casi si utilizzano i tipi stringa (**REG_SZ**), binario (**REG_BINARY**) e valore Double Word, un numero intero a 32 bit e senza segno (**REG_DWORD**). Sono questi infatti i tipi di dati che è possibile inserire nel Registry utilizzando il programma Regedit. Questa classe non implementa i tipi di dati

REG_LINK, **REG_RESOURCE_LIST** e **REG_DWORD_BIG_ENDIAN**.

```
61. Private lngKeyValue As Long
62. Private lngKeySecurity As Long
63.
```

Gli unici due membri privati interni sono due valori di tipo Long: il primo identifica l'handle alla chiave aperta (vedi l'enumerazione ***ChiaviPrincipali***) e verrà utilizzato per la maggior parte delle operazioni. L'altro valore rappresenta invece un indice di sicurezza come indicato dall'enumerazione ***ChiaviSecurity***.

```
64. Private Sub Class_Initialize()
65.   lngKeyValue = 0
66. End Sub
67.
68. Private Sub Class_Terminate()
69.   Call RegCloseKey(lngKeyValue)
70. End Sub
71.
```

Giusto per sicurezza, all'istanza della classe la variabile **lngKeyValue** viene posta uguale a 0 per indicare che nessuna chiave è stata ancora aperta. Alla deallocazione dell'istanza viene comunque effettuata la chiusura della chiave **lngKeyValue**. È importante ricordarsi di chiudere le chiavi quando esse non sono più necessarie e, come vedremo in seguito, non chiudere mai le chiavi delle quali non è stata effettuata la diretta apertura.

Segue parte 2 >>

Fibia FBI
1 Aprile 2002
Corretto il 20 Settembre 2002

Torna all'indice degli HowTo

Utilizzare il registro di Windows (quinta parte)

http://www.vbsimple.net/howto/ht_044_5.htm

Difficoltà: 5 / 5

[<< Continua dalla parte 4]

Concludiamo questo lungo articolo con le ultime due funzioni della classe: **SeparaValore** e **Importa**: la prima servirà per estrarre da un file di testo il corretto nome e contenuto di un valore; la seconda si occuperà di leggere i dati da un file di testo compatibile con Regedit ed inserirli nel registro nella corretta posizione.

```
440. Private Function SeparaValore(ByVal Dati As String, ByRef NomeValore As String) As
     String
441.   Dim Conta As Integer
442.   Conta = InStr(2, Dati, """")
443.   Do While Conta > 0
444.     If Mid$(Dati, Conta - 1, 2) <> "\""" Then Exit Do
445.     Conta = InStr(Conta + 1, Dati, """")
446.   Loop
447.   If Left$(Dati, 1) = "@" Then Conta = 1
448.   NomeValore = Left$(Dati, Conta)
449.   NomeValore = Replace(NomeValore, "\\", "\")
450.   NomeValore = Replace(NomeValore, "\r", vbCr)
451.   NomeValore = Replace(NomeValore, "\n", vbLf)
452.   NomeValore = Replace(NomeValore, "\""", """")
453.   SeparaValore = Mid$(Dati, Conta + 2)
454.   SeparaValore = Replace(SeparaValore, "\\", "\")
455.   SeparaValore = Replace(SeparaValore, "\r", vbCr)
456.   SeparaValore = Replace(SeparaValore, "\n", vbLf)
457.   SeparaValore = Replace(SeparaValore, "\""", """")
458. End Function
459.
```

Semplice quanto banale, la funzione **SeparaValore** ricostruisce il corretto nome e contenuto di un valore dal formato proveniente da file (ovvero alterato dalla procedura di esportazione o dal programma Regedit). La ricostruzione terrà conto che all'interno del nome di un valore possono esserci anche quei simboli particolari quali le virgolette o il segno di uguaglianza e pertanto la separazione tra nome e contenuto del valore deve tenerne conto prima (righe 442-446), ed eliminare quelle alterazioni (righe 447-457).

```
460. Public Function Importa(ByVal NomeFile As String) As Boolean
461.   Dim FileNR As Integer
462.   Dim LineBuffer As String
463.   Dim Buffer As Variant
464.   Dim oldKey As Long
465.   Dim newKey As Long
466.   Dim Conta As Integer
467.   Dim Buffer2() As Byte
468.   Dim TipoDati As TipoValoriRegistro
469.   FileNR = FreeFile
470.   On Error Resume Next
471.   Open NomeFile For Input As FileNR
472.   If Err.Number <> 0 Then
473.     MsgBox "Errore durante la preparazione all'importazione!", vbCritical +
```



```
     vbOKOnly, "FBIRegistry"
474.      Importa = False
475.      Close FileNR
476.      Exit Function
477.    End If
478.    Line Input #FileNR, LineBuffer
479.    If UCase$(LineBuffer) <> "REGEDIT4" Then
480.      MsgBox "Il formato del file da importare non è corretto!", vbCritical +
     vbOKOnly, "FBIRegistry"
481.      Importa = False
482.      Exit Function
483.    End If
```

Prima di iniziare l'elaborazione vera e propria saranno effettuati quei controlli d'obbligo quale l'accessibilità al file ed il controllo della sua intestazione (***REGEDIT4***). Nella mancanza di uno di questi due requisiti l'elaborazione verrà interrotta con un messaggio di errore (righe 472-483).

```
484.    oldKey = lngKeyValue
485.    lngKeyValue = 0
486.    Do While Not EOF(FileNR)
487.      Line Input #FileNR, LineBuffer
488.      LineBuffer = Trim$(LineBuffer)
489.      Buffer = ""
490.      Do While Right$(LineBuffer, 1) = "\"
491.        Buffer = Buffer & Left$(LineBuffer, Len(LineBuffer) - 1)
492.        Line Input #FileNR, LineBuffer
493.        LineBuffer = Trim$(LineBuffer)
494.      Loop
```

Verrà quindi letto il contenuto del file linea per linea (riga 487). Se i dati letti dal file termineranno sulla destra con un `"\"` sarà segno di un dato binario spezzato su più righe e pertanto sarà necessario ricostruire l'intero valore (righe 490-494) leggendo una per una le righe successive.

```
495.      If LineBuffer <> "" Then
496.        Buffer = Buffer & LineBuffer
497.        If Left$(Buffer, 1) = "[" Then
498.          Buffer = Mid$(Buffer, 2, Len(Buffer) - 2)
499.          Conta = InStr(1, Buffer, "\")
500.          LineBuffer = Buffer
501.          If Conta > 0 Then LineBuffer = UCase$(Left$(Buffer, Conta - 1))
502.          Select Case LineBuffer
503.            Case "HKEY_CLASSES_ROOT": newKey = HKEY_CLASSES_ROOT
504.            Case "HKEY_CURRENT_CONFIG": newKey = HKEY_CURRENT_CONFIG
505.            Case "HKEY_CURRENT_USER": newKey = HKEY_CURRENT_USER
506.            Case "HKEY_DYN_DATA": newKey = HKEY_DYN_DATA
507.            Case "HKEY_LOCAL_MACHINE": newKey = HKEY_LOCAL_MACHINE
508.            Case "HKEY_PERF_ROOT": newKey = HKEY_PERF_ROOT
509.            Case "HKEY_PERFORMANCE_DATA": newKey = HKEY_PERFORMANCE_DATA
510.            Case "HKEY_USERS": newKey = HKEY_USERS
511.            Case Else
512.              MsgBox "La chiave principale non è una chiave di sistema.", vbCritical
     + vbOKOnly, "FBIRegistry"
513.              Importa = False
514.              Close FileNR
515.              Exit Function
516.          End Select
```

Ottenuta quindi una riga completa di dati dal file sarà necessario innanzitutto verificare se si tratti di una chiave oppure di un valore. Le chiavi sono identificate da una coppia di parentesi quadre agli estremi nella forma `"[chiave]"`. Pertanto i dati che iniziano per `"["` identificano una chiave e come tale deve essere trattata.

Verranno quindi estratte la chiave di sistema e la sottochiave rispetto a questa e salvata nella variabile **newKey** (righe 498-516).

```
517.          Call RegCloseKey(lngKeyValue)
518.          lngKeyValue = 0
519.          Call RegCreateKeyEx(newKey, Mid$(Buffer, Len(LineBuffer) + 2), ByVal 0&,
      vbNullChar, REG_OPTION_NON_VOLATILE, lngKeySecurity, ByVal 0&, lngKeyValue, ByVal
      0&)
```

Sarà adesso necessario aprire o creare la nuova chiave ottenuta. La funzione *RegCreateKeyEx* crea la chiave se non esiste ma provvede anche la sua apertura nel caso essa esista. Prima di aprire la chiave però, verrà chiusa la chiave eventualmente aperta in precedenza tramite *RegCloseKey*.

```
520.        Else
521.          Conta = InStr(1, Buffer, "=")
522.          If Conta > 0 Then
523.            Buffer = SeparaValore(Buffer, LineBuffer)
524.            If LineBuffer = "@" Then LineBuffer = ""
525.            LineBuffer = DeQuote(LineBuffer)
526.            If Left$(Buffer, 1) = """" Then
527.              TipoDati = REG_SZ
528.              Buffer = DeQuote(Buffer)
529.              Buffer2 = StrConv(Buffer, vbFromUnicode) & vbNullChar
```

Se invece il dato estratto dal file non è una chiave, con grossissima probabilità si tratterà di un valore dell'ultima chiave aperta. Alla riga 523 vengono estratti nome (nella variabile **LineBuffer**) e contenuto (nella variabile di ritorno **Buffer**) del valore recuperato da file. Saranno in seguito fatti quei minimi aggiustamenti (righe 524-525) per ricostruire il nome valore nella maniera corretta.

Soltanto adesso potrà essere fatta l'analisi dei dati recuperati per determinare il tipo di dati trattati. Se il contenuto di tale valore dovesse iniziare con le virgolette ci troveremmo in presenza di un valore di tipo **REG_SZ** la cui conversione in array di bytes risulta molto semplice (righe 527-529).

```
530.            ElseIf UCase$(Left$(Buffer, 3)) = "HEX" Then
531.              TipoDati = REG_BINARY
532.              If UCase$(Left$(Buffer, 4)) = "HEX(" Then
533.                TipoDati = CByte(Mid$(Buffer, 5, 1))
534.              End If
535.              Conta = InStr(1, Buffer, ":")
536.              Buffer = Mid$(Buffer, Conta + 1)
537.              ReDim Buffer2(Ceil(Len(Buffer) / 3)) As Byte
538.              For Conta = 0 To UBound(Buffer2) - 1
539.                Buffer2(Conta) = CByte("&H" & Mid$(Buffer, Conta * 3 + 1, 2))
540.              Next Conta
```

Diversamente se il contenuto di tali dati inizia con la stringa `"Hex"` ci troveremmo in presenza di dati binari (righe 530-531). Un'eventuale parentesi successiva alla stringa `"Hex"` indicherà il tipo di dati binari (righe 532-534).

Sarà quindi allocato un buffer di bytes in base all'ampiezza dei dati binari recuperati. L'ampiezza è calcolata come arrotondamento per eccesso (*Ceil*) della lunghezza del buffer diviso per 3. Il numero tre sta ad indicare 2 bytes per ogni cifra binaria ed un byte per la virgola separatrice. Tali bytes saranno poi riconvertiti in numeri decimali ed inseriti nel buffer binario.


```
541.           ElseIf UCase$(Left$(Buffer, 5)) = "DWORD" Then
542.             TipoDati = REG_DWORD
543.             Buffer = Mid$(Buffer, 7)
544.             ReDim Buffer2(Ceil(Len(Buffer) / 2)) As Byte
545.             For Conta = 1 To Len(Buffer) Step 2
546.               Buffer2(Conta \ 2) = CByte("&H" & Mid$(Buffer, Len(Buffer) - Conta,
     2))
547.             Next Conta
548.           Else
549.             MsgBox "Errore nell'importazione del valore """ & LineBuffer & "=" &
     Buffer, vbCritical + vbOKOnly, "FBIRegistry"
550.             Importa = False
551.             Exit Function
552.           End If
```

Se invece il tipo di dati estratto non è neanche quello binari si tratterà necessariamente del tipo di dati **DWORD** oppure di un caso di errore. Sarà quindi controllata la presenza della stringa "DWORD" e nel caso positivo verrà ricostruito il valore **DWORD** (righe 541-547).

Come ultimissima ipotesi, se non si tratta né di stringhe, né di dati binari e né tantomeno di valori DWORD abbiamo dinanzi un caso di errore che non può essere gestito. Sarà generato un avviso e la procedura di importazione verrà interrotta (righe 548-551).

```
553.           Call RegSetValueEx(lngKeyValue, LineBuffer, ByVal 0&, TipoDati, Buffer2
     (0), UBound(Buffer2))
554.         Else
555.           MsgBox "Il valore presenta una forma anomala.", vbCritical + vbOKOnly,
     "FBIRegistry"
556.           Importa = False
557.           Exit Function
558.         End If
559.       End If
560.     End If
561.   Loop
562.   Call RegCloseKey(lngKeyValue)
563.   Close FileNR
564.   lngKeyValue = oldKey
565.   Importa = True
566. End Function
```

Il raggiungimento della riga 553 dovrebbe indicare nessuna interruzione e quindi un buffer caricato con i dati corretti e nella forma desiderata. La variabile **TipoDati** dovrebbe invece contenere il tipo di dati desiderato. Saranno quindi salvati tali dati sul registro tramite *RegSetValueEx*.

Alla riga 522 abbiamo verificato la presenza di un simbolo "=" ad indicare la separazione tra nome del valore e suo contenuto. Se tale simbolo non dovesse essere affatto presente avremmo dinanzi un altro emblematico caso di errore; sarà generato un avviso e l'esecuzione verrà interrotta (righe 554-557).

Al termine dell'importazione di tutti i valori sarà chiusa la chiave aperta e si potrà quindi procedere all'importazione della chiave successiva, fino alla fine del file, segnata anche dalla sua chiusura.

Questo lunghissimo articolo si conclude qui. Ci sarebbe molto altro da dire ma risulterebbe troppo pesante. La classe sviluppata si presenta parecchio complessa e lunga ma altrettanto

semplice e ben fatta.

Nel progetto da scaricare è presente anche un semplicissimo esempio a scopo dimostrativo delle principali funzioni della classe FBIRegistry, tutta da scoprire.
La classe è stata testata soltanto su Windows 9x utilizzando un file di registro di oltre 10 MB e svolge egregiamente il suo compito, tanto da riprodurre gli stessi risultati del programma Regedit (esclusa la velocità, ovviamente, limite invalicabile di Visual Basic).

Fibia FBI
1 Aprile 2002
Corretto il 20 Settembre 2002

Torna all'indice degli HowTo

Utilizzare il registro di Windows (seconda parte)

http://www.vbsimple.net/howto/ht_044_2.htm

Difficoltà: 5 / 5

[<< Continua dalla parte 1]

Dopo l'introduttiva presentazione delle dichiarazioni della classe passiamo direttamente alla fase implementativa; vedremo in ordine le proprietà pubbliche della classe, i suoi metodi pubblici ed infine quelli privati. L'ordine seguito è quello esclusivamente alfabetico per i tre gruppi; pertanto a volte saranno utilizzate funzioni descritte soltanto in seguito.

```
72. Public Property Get Chiave() As Long
73.   Chiave = lngKeyValue
74. End Property
75.
76. Public Property Let Chiave(ByVal newChiave As Long)
77.   Call RegOpenKeyEx(newChiave, vbNullString, ByVal 0&, lngKeySecurity, newChiave)
78.   Call RegCloseKey(lngKeyValue)
79.   lngKeyValue = newChiave
80. End Property
81.
```

La proprietà **Chiave** di lettura e scrittura restituisce ed imposta l'handle della chiave utilizzata dalla classe assegnando il contenuto del membro interno. L'operazione di scrittura della proprietà è però leggermente più complessa: prima di assegnare il nuovo valore alla variabile ***lngKeyValue*** è necessario riaprire la nuova chiave (riga 77) e chiudere quella precedente (riga 78).

Questo perché Windows tiene conto del numero di volte che la chiave è stata aperta. Se due istanze aprono la stessa chiave, ed una delle due istanze ne richiede la chiusura, l'handle dell'altra istanza non verrà invalidato ed entrambe le chiavi chiuse.
È pertanto fondamentale chiudere soltanto la chiave che l'istanza ha aperto e riaprire la nuova chiave ogni volta che un'istanza ne usufruisce. Questo garantisce che la chiave non venga chiusa da un'altra istanza o da un altro processo.

```
82. Public Property Get NumeroSottoChiavi() As Long
83.   Call RegQueryInfoKey(lngKeyValue, vbNullChar, ByVal 0&, ByVal 0&,
    NumeroSottoChiavi, ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&,
    ByVal 0&)
84. End Property
85.
86. Public Property Get NumeroValori() As Long
87.   Call RegQueryInfoKey(lngKeyValue, vbNullChar, ByVal 0&, ByVal 0&, ByVal 0&, ByVal
    0&, ByVal 0&, NumeroValori, ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&)
88. End Property
89.
```

Seguono due proprietà in sola lettura che restituiscono rispettivamente il numero di sottochiavi ed il numero di valori della chiave aperta. Entrambe le proprietà utilizzano la

funzione API *RegQueryInfoKey*.

```
 90. Public Property Get Security() As ChiaviSecurity
 91.   Security = lngKeySecurity
 92. End Property
 93.
 94. Public Property Let Security(ByVal newSecurity As ChiaviSecurity)
 95.   lngKeySecurity = newSecurity
 96. End Property
 97.
```

La proprietà **Security** consente di recuperare ed impostare un valore relativo ai permessi concessi. Trascurabile su Windows 95/98/ME.

```
 98. Public Property Get Valore(Optional ByVal NomeValore As String, Optional ByRef
     TipoDati As TipoValoriRegistro) As Variant
 99.   Dim AmpiezzaBuffer As Long
100.   Dim Buffer() As Byte
101.   TipoDati = REG_NONE
102.   Call RegQueryValueEx(lngKeyValue, NomeValore, ByVal 0&, TipoDati, ByVal 0&,
     AmpiezzaBuffer)
103.   If AmpiezzaBuffer > 0 Then
104.     ReDim Buffer(AmpiezzaBuffer - 1)
105.     Call RegQueryValueEx(lngKeyValue, NomeValore, ByVal 0&, ByVal 0&, Buffer(0),
     AmpiezzaBuffer)
106.   End If
```

La proprietà **Valore** si presenta un pochino più complessa delle precedenti. L'operazione di lettura recupera il contenuto dei valori: quello predefinito della chiave se non viene specificato il nome del valore, o quello indicato se ne viene specificato il nome.

Sarà anche possibile recuperare con la stessa proprietà il tipo di dato passando una variabile come secondo argomento. Al ritorno della funzione tale variabile conterrà il tipo di dati del valore richiesto.

Alla riga 102 sono richiesti il tipo di dato del valore e l'ampiezza del contenuto del valore mediante la funzione *RegQueryValueEx*. Se il dato può essere recuperato, la variabile **AmpiezzaBuffer** conterrà la dimensione in bytes del buffer da allocare.

Tale buffer sarà effettivamente allocato alla riga 104 ed i dati potranno quindi essere recuperati richiamando nuovamente la funzione *RegQueryValueEx* e fornendo il puntatore al buffer precedentemente allocato.

```
107.   Select Case TipoDati
108.     Case REG_SZ
109.       Valore = Left$(StrConv(Buffer, vbUnicode), AmpiezzaBuffer - 1)
110.     Case REG_NONE, REG_BINARY, REG_EXPAND_SZ, REG_MULTI_SZ
111.       Valore = Buffer
112.     Case REG_DWORD, REG_DWORD_LITTLE_ENDIAN
113.       If UBound(Buffer) = 3 Then
114.         Valore = CDbl(Buffer(0) + Buffer(1) * 256& + Buffer(2) * 65536) + Buffer(3)
     * 16777216#
115.       Else
116.         Valore = Buffer
117.       End If
118.     Case REG_DWORD_BIG_ENDIAN, REG_LINK, REG_RESOURCE_LIST
119.       MsgBox "Tipo di dati non implementato [Valore Get]!", vbCritical + vbOKOnly,
     "FBIRegistry"
120.   End Select
121.   If AmpiezzaBuffer = 0 Then Valore = Null
122.   Erase Buffer
```



```
123. End Property
124.
```

Avendo recuperato il contenuto del valore ed il suo tipo di dati, sarà quindi possibile effettuare la conversione nel formato più convenzionale: stringa per il tipo **REG_SZ**, numero intero double per il tipo **REG_DWORD** o matrice di byte per gli altri tipi di dato. Per quanto riguarda la conversione in stringa verrà effettuata la conversione da formato ANSI ad Unicode (riga 109). La conversione in numero double effettuerà un controllo in più alla riga 113: sebbene il processo di conversione è semplice (riga 114), esistono all'interno del registro alcuni valore memorizzati in maniera errata, ovvero con un numero di bytes minore o maggiore dei normali quattro che compongono un valore DWORD. Per tali casi sarà semplicemente restituita la matrice di dati senza effettuare la conversione (riga 116).

Se il valore richiesto non esiste all'interno del registro oppure non è impostato alcun valore per esso, la dimensione del buffer sarà uguale a 0 e sarà pertanto restituito un valore ***Null*** (riga 121).

```
125. Public Property Let Valore(ByVal NomeValore As String, Optional ByRef TipoDati As
     TipoValoriRegistro = REG_SZ, ByVal newValore As Variant)
126.   Dim AmpiezzaBuffer As Long
127.   Dim Buffer() As Byte
128.   If (TipoDati < REG_NONE) Or (TipoDati > REG_RESOURCE_LIST) Then TipoDati = REG_SZ
129.   Select Case TipoDati
130.     Case REG_SZ
131.       Buffer = StrConv(newValore, vbFromUnicode) & vbNullChar
```

L'ultima proprietà è l'operazione di scrittura sul valore del registro. Come la precedente consente opzionalmente la specifica di un nome di valore e di un tipo di dati che se non specificato assume il significato di stringa (**REG_SZ**). Alla riga 128 viene effettuato un controllo d'obbligo sul tipo di dato specificato.

Se il dato da scrivere è di tipo stringa esso sarà semplicemente convertito da Unicode ad ANSI e trasferito nella variabile binaria **Buffer**.

```
132.     Case REG_BINARY, REG_NONE, REG_EXPAND_SZ, REG_MULTI_SZ
133.       If (VarType(newValore) And vbArray) = vbArray Then
134.         ReDim Buffer(0) As Byte
135.         If Not IsNull(newValore) Then ReDim Buffer(UBound(newValore) + 1) As Byte
136.         Buffer(UBound(Buffer)) = 0
137.         Select Case (VarType(newValore) Xor vbArray)
138.           Case vbByte
139.             Buffer = newValore
140.             If Not IsNull(newValore) Then ReDim Preserve Buffer(UBound(newValore) +
     1) As Byte
141.             Buffer(UBound(Buffer)) = 0
142.           Case vbInteger, vbLong, vbDouble, vbVariant
143.             For AmpiezzaBuffer = LBound(newValore) To UBound(newValore)
144.               Buffer(AmpiezzaBuffer) = (newValore(AmpiezzaBuffer) And 255)
145.             Next AmpiezzaBuffer
146.           Case Else
147.             MsgBox "Tipo di dati non supportato [Valore Let]!", vbCritical +
     vbOKOnly, "FBIRegistry"
148.             Exit Property
149.         End Select
```

La situazione si complica nel caso dei tipi di dati binari specificato alla riga 132. Essi dovranno essere convertiti in maniera idonea in una matrice. L'operazione sarebbe uno scherzo se il tipo di dati non fosse Variant e come tale Visual Basic si rifiuterà di trattarlo

o trasformarlo direttamente in matrice di byte, a meno che il suo sottotipo non sia Byte.

Sarà quindi verificato alla riga 133 che il nuovo valore passato sia una matrice: in tal caso verrà allocata una nuova matrice di bytes di nome **Buffer** della dimensione dell'altra matrice ed il contenuto verrà quindi ricopiato nella matrice **Buffer**. Sarà generato un errore se il sottotipo dei dati non è numerico (riga 147).

```
150.       ElseIf IsNull(newValore) = False Then
151.         MsgBox "I dati binari devono essere in una matrice!", vbCritical +
     vbOKOnly, "FBIRegistry"
152.         Exit Property
153.       Else
154.         ReDim Buffer(0) As Byte
155.       End If
156.     Case REG_DWORD, REG_DWORD_LITTLE_ENDIAN
157.       If IsNull(newValore) = False Then
158.         If IsArray(newValore) = False Then
159.           ReDim Buffer(4) As Byte
160.           Buffer(0) = (newValore And 255)
161.           Buffer(1) = (newValore \ 256) And 255
162.           Buffer(2) = (newValore And &H7FFFFF00) \ 65536 And 255
163.           Buffer(3) = ((newValore And &HFF000000) \ 16777216) And 255
164.         Else
165.           ReDim Buffer(UBound(newValore) + 1) As Byte
166.           For AmpiezzaBuffer = LBound(newValore) To UBound(newValore)
167.             Buffer(AmpiezzaBuffer) = (newValore(AmpiezzaBuffer) And 255)
168.           Next AmpiezzaBuffer
169.         End If
170.       Else
171.         ReDim Buffer(0) As Byte
172.       End If
```

Sarà generato un avviso anche nel caso in cui i dati passati non siano in una matrice ed il tipo richiesto sia comunque di dati binari (riga 151). Nell'ultimo caso che il valore passato sia un valore Nullo sarà generata una matrice vuota (riga 154).

Se invece il tipo di dati richiesto è il **REG_DWORD** sarà verificato che i dati non siano una matrice. Nel caso che non lo fossero dovrà quindi essere calcolato il nuovo valore ed assegnarlo ad una matrice di 4 bytes (righe 159-163). Se i dati sono invece in una matrice tali dati saranno ricopiati nel nuovo buffer (righe 165-168).

```
173.     Case REG_DWORD_BIG_ENDIAN, REG_LINK, REG_RESOURCE_LIST
174.       MsgBox "Tipo di dati non implementato [Valore Let]!", vbCritical + vbOKOnly,
     "FBIRegistry"
175.       Exit Property
176.   End Select
177.   If IsNull(newValore) And (TipoDati = REG_SZ) Then
178.     Call RegDeleteValue(lngKeyValue, NomeValore)
179.   Else
180.     Call RegSetValueEx(lngKeyValue, NomeValore, ByVal 0&, TipoDati, Buffer(0),
     UBound(Buffer))
181.   End If
182.   Erase Buffer
183. End Property
184.
```

Tutti gli altri tipi di dati non sono supportati (righe 173-175).
Prima di procedere con la scrittura dei dati richiesti sarà effettuato un ultimo controllo: se il dato da scrivere è un valore Null ed il tipo richiesto è la stringa **REG_SZ** il valore non sarà scritto affatto, anzi sarà cancellato del tutto il valore (riga 178) tramite funzione API *RegDeleteValue*. Nel caso contrario sarà effettuata la normale scrittura del buffer tramite la

funzione *RegSetValueEx* (riga 180).

Segue parte 3 >>

Fibia FBI
1 Aprile 2002
Corretto il 20 Settembre 2002


Torna all'indice degli HowTo

Utilizzare il registro di Windows (terza parte)

http://www.vbsimple.net/howto/ht_044_3.htm

Difficoltà: 5 / 5

[<< Continua dalla parte 2]

Quella che segue è la parte più complessa dell'intero articolo: saranno trattati i metodi pubblici e privati della nostra classe *clsFBIRegistry*.

```
185. Public Function ApriChiave(ByVal ChiavePrincipale As ChiaviPrincipali, ByVal
     SottoChiave As String) As Long
186.   Dim oldKey As Long
187.   oldKey = lngKeyValue
188.   If ChiavePrincipale = CHIAVE_APERTA Then ChiavePrincipale = lngKeyValue
189.   lngKeyValue = 0
190.   Call RegOpenKeyEx(ChiavePrincipale, SottoChiave, ByVal 0&, lngKeySecurity,
     lngKeyValue)
191.   ApriChiave = lngKeyValue
192.   Call RegCloseKey(oldKey)
193. End Function
194.
195. Public Sub ChiudiChiave()
196.   Call RegCloseKey(lngKeyValue)
197.   lngKeyValue = 0
198. End Sub
199.
```

Le prime due funzioni sono le più scontate: **ApriChiave** effettua l'apertura di una chiave o di una sottochiave, mentre **ChiudiChiave** chiude la chiave aperta. Per la prima funzione potrà essere fornito l'handle ad una qualsiasi chiave aperta (da un'altra istanza o processo) oppure una chiave di sistema (come definito dall'enumerazione **ChiaviPrincipali**). Nel caso venisse utilizzato il valore **CHIAVE_APERTA** sarà utilizzato l'handle della chiave aperta dall'istanza della nostra classe. Il parametro sottochiave invece identicherà la sottochiave da apire rispetto alla chiave specificata.

La funzione **ChiudiChiave** invece effettua semplicemente la chiusura della chiave e l'azzeramento dell'handle **lngKeyValue**.

```
200. Public Function CreaChiave(ByVal ChiavePrincipale As ChiaviPrincipali, ByVal
     SottoChiave As String, Optional ByVal NonCambiare As Boolean = False) As Long
201.   Dim newKeyValue As Long
202.   If ChiavePrincipale = CHIAVE_APERTA Then ChiavePrincipale = lngKeyValue
203.   Call RegCreateKeyEx(ChiavePrincipale, SottoChiave, ByVal 0&, vbNullChar,
     REG_OPTION_NON_VOLATILE, lngKeySecurity, ByVal 0&, newKeyValue, ByVal 0&)
204.   If NonCambiare = False Then
205.     Call RegCloseKey(lngKeyValue)
206.     lngKeyValue = newKeyValue
207.   End If
208.   CreaChiave = newKeyValue
209. End Function
210.
```


In maniera analoga alla funzione vista in precedenza, la funzione **CreaChiave** consente la creazione e l'apertura di una sottochiave. Se la sottochiave richiesta non esiste essa verrà creata ed aperta. Il funzionamento è analogo alla funzione **ApriChiave** ma il parametro aggiuntivo **NonCambiare** consente di effettuare le due succitate operazioni senza tuttavia chiudere e cambiare la chiave utilizzata dall'istanza, ed indicata dalla proprietà **Chiave**.

Questa possibilità si rivela utile quando si desidera creare una nuova sottochiave, ottenerne l'handle (come valore di ritorno della funzione) senza però chiudere la chiave sulla quale l'istanza sta lavorando.

```
211. Public Function ElencaChiave(ByVal ChiavePrincipale As ChiaviPrincipali, ByVal
     Indice As Long) As String
212.   Dim Buffer As String
213.   Dim Lunghezza As Long
214.   If ChiavePrincipale = CHIAVE_APERTA Then ChiavePrincipale = lngKeyValue
215.   Buffer = String$(255, 0)
216.   Lunghezza = 255
217.   Call RegEnumKeyEx(ChiavePrincipale, Indice, Buffer, Lunghezza, ByVal 0&,
     vbNullString, ByVal 0&, ByVal 0&)
218.   ElencaChiave = Left$(Buffer, Lunghezza)
219. End Function
220.
```

La funzione **ElencaChiave** consente di recuperare il nome di una sottochiave in base al suo ordine, ad esempio la prima, la seconda o la terza sottochiave della chiave aperta. Viene utilizzata a tale scopo la funzione API *RegEnumKeyEx*.

```
221. Public Function ElencaChiavi(ByVal ChiavePrincipale As ChiaviPrincipali) As Variant
222.   Dim Elenco() As String
223.   Dim Conta As Long
224.   Dim Buffer As String
225.   If ChiavePrincipale = CHIAVE_APERTA Then ChiavePrincipale = lngKeyValue
226.   Call RegQueryInfoKey(ChiavePrincipale, vbNullChar, ByVal 0&, ByVal 0&, Conta,
     ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&)
227.   If Conta > 0 Then
228.     ReDim Elenco(Conta - 1) As String
229.     Buffer = String$(255, 0)
230.     Conta = 0
231.     Do While RegEnumKeyEx(ChiavePrincipale, Conta, Buffer, Len(Buffer), ByVal 0&,
     vbNullString, ByVal 0&, ByVal 0&) = 0
232.       Elenco(Conta) = Left$(Buffer, InStr(1, Buffer, Chr$(0), vbBinaryCompare) - 1)
233.       Conta = Conta + 1
234.       Buffer = String$(255, 0)
235.     Loop
236.     ElencaChiavi = Elenco
237.   Else
238.     ElencaChiavi = Null
239.   End If
240. End Function
241.
```

In maniera analoga alla funzione precedente, la funzione **ElencaChiavi** restituisce un array contenente i nomi di tutte le sottochiavi presenti all'interno della chiave specificata. Se nessuna sottochiave è presente, restituirà valore Null. Utilizza la funzione *RegQueryInfoKey* allo stesso modo della proprietà NumeroSottochiavi per recuperare il numero complessivo delle sottochiavi ed allocare così il buffer necessario e poi ne estrae uno per uno i nomi utilizzando la funzione *RegEnumKeyEx* nella stessa maniera della funzione **ElencaChiave**.

```
242. Public Function ElencaValore(ByVal ChiavePrincipale As ChiaviPrincipali, ByVal
     Indice As Long) As String
```



```
243.   Dim Buffer As String
244.   Dim Lunghezza As Long
245.   If ChiavePrincipale = CHIAVE_APERTA Then ChiavePrincipale = lngKeyValue
246.   Buffer = String$(255, 0)
247.   Lunghezza = 255
248.   Call RegEnumValue(ChiavePrincipale, Indice, Buffer, Lunghezza, ByVal 0&, ByVal
     0&, ByVal 0&, ByVal 0&)
249.   ElencaValore = Left$(Buffer, Lunghezza)
250. End Function
251.
```

La funzione **ElencaValore** consente di recuperare il nome di un valore specifico, di cui si possiede l'indice, all'interno della chiave indicata. Il funzionamento è molto simile a quello della funzione **ElencaChiave** con l'unica differenza che utilizza la funzione *RegEnumValue* per recuperare l'informazione dal registro.

```
252. Public Function ElencaValori(ByVal ChiavePrincipale As ChiaviPrincipali) As Variant
253.   Dim Elenco() As String
254.   Dim Conta As Long
255.   Dim Buffer As String
256.   If ChiavePrincipale = CHIAVE_APERTA Then ChiavePrincipale = lngKeyValue
257.   Call RegQueryInfoKey(lngKeyValue, vbNullChar, ByVal 0&, ByVal 0&, ByVal 0&, ByVal
     0&, ByVal 0&, Conta, ByVal 0&, ByVal 0&, ByVal 0&, ByVal 0&)
258.   If Conta > 0 Then
259.     ReDim Elenco(Conta - 1)
260.     Buffer = String$(255, 0)
261.     Conta = 0
262.     Do While RegEnumValue(ChiavePrincipale, Conta, Buffer, Len(Buffer), ByVal 0&,
     ByVal 0&, ByVal 0&, ByVal 0&) = 0
263.       Elenco(Conta) = Left$(Buffer, InStr(1, Buffer, Chr$(0), vbBinaryCompare) - 1)
264.       Conta = Conta + 1
265.       Buffer = String$(255, 0)
266.     Loop
267.     ElencaValori = Elenco
268.   Else
269.     ElencaValori = Null
270.   End If
271. End Function
272.
```

E come la sua parente, la funzione **ElencaValori** restituisce un array contenente i nomi di tutti i valori presenti all'interno della chiave specificata. Restituisce invece Null in assenza di valori all'interno della chiave.

```
273. Public Function EliminaChiave(ByVal ChiavePrincipale As ChiaviPrincipali, Optional
     ByVal SottoChiave As String) As Boolean
274.   If ChiavePrincipale = CHIAVE_APERTA Then ChiavePrincipale = lngKeyValue
275.   EliminaChiave = (RegDeleteKey(ChiavePrincipale, SottoChiave) = 0)
276. End Function
277.
278. Public Function EliminaValore(Optional ByVal NomeValore As String = "") As Boolean
279.   EliminaValore = (RegDeleteValue(lngKeyValue, NomeValore) = 0)
280. End Function
281.
```

Le due funzioni di eliminazione: **EliminaChiave** ed **EliminaValore** consentono l'eliminazione rispettivamente di una chiave e di un valore.

Tralasciamo per il momento l'ordine alfabetico e, saltando la funzione Esporta, passiamo direttamente alla funzione **ForzaAggiornamentoChiave**. Vista la loro complessità, le due funzioni di esportazione ed importazione saranno trattate nella prossima parte.

```
282. Public Sub ForzaAggiornamentoChiave()
```



```
283.    Call RegFlushKey(lngKeyValue)
284.  End Sub
285.
```

La sua utilità è dubbia e potrebbe generare sospetti sul corretto funzionamento della classe. Nel momento in cui viene richiesta la modifica di un valore nel registro i dati sono effettivamente alterati ma non immediatamente scritti su disco. Se ipoteticamente, un secondo dopo un'operazione di scrittura sul registro, andasse via la luce, al riavvio del computer il valore potrebbe non essere stato ancora scritto.

Questo perché Windows utilizza un sistema di cache e di scrittura ritardata (*lazy write*) che non impegna troppo il computer in lente operazioni di scrittura. Essa seguirà infatti nei momenti più liberi del computer.

La funzione **ForzaAggiornamentoChiave** e quindi *RegFlushKey* obbligano il sistema operativo a scrivere su disco tutte le modifiche relative alla chiave specificata, garantendo l'immediata scrittura su disco ma rallentando pericolosamente la velocità del sistema. In linea generale questa funzione non è necessaria ma in rarissimi casi può richiedersi utile. La funzione *RegFlushKey* viene chiamata dal sistema operativo sulle chiavi di sistema al momento dello spegnimento della macchina per assicurare la corretta scrittura dei dati su disco.

Prima di vedere le ultime due funzioni pubbliche relative all'esportazione ed all'importazione dei dati, diamo un rapido sguardo ad alcune funzioni private utilizzate in altre parti del codice.

```
286.  Private Function DeQuote(ByVal Stringa As String) As String
287.    If Left$(Stringa, 1) = """" Then Stringa = Mid$(Stringa, 2)
288.    If Right$(Stringa, 1) = """" Then Stringa = Left$(Stringa, Len(Stringa) - 1)
289.    DeQuote = Stringa
290.  End Function
291.
292.  Private Function Ceil(ByVal Numero As Double) As Double
293.  Ceil = Fix(Numero)
294.  If Numero > Fix(Numero) Then Ceil = Ceil + 1
295.  End Function
296.
```

La funzione **DeQuote** consente di rimuovere l'eventuali virgolette (") alla sinistra ed alla destra di una stringa. Non rimuovono tutte le virgolette ma soltanto un singolo paio. Viene utilizzata per recuperare il nome reale dei valori durante la fase di importazione da file.

La funzione **Ceil** prende il nome dalla funzione analoga del linguaggio JavaScript e restituisce il numero arrotodonato **sempre** per eccesso. È utilizzata per determinare l'ampiezza del buffer durante il processo di importazione.

```
297.  Private Function HexDouble(ByVal Numero As Double, Optional Padding As Integer) As
      String
298.    Dim HighPart As Double
299.    Numero = Fix(Numero)
300.    HighPart = Fix(Numero / 65536)
301.    Numero = Numero - HighPart * 65536
302.    HexDouble = Hex$(HighPart) & Right$("0000" & Hex$(Numero), 4)
303.    If Padding > 0 Then HexDouble = Right$(String$(Padding, "0") & HexDouble,
      Padding)
```



```
304.  End Function
305.
```

Una funzione un po' unusuale è la **HexDouble** ed in un linguaggio quale il C non avrebbe probabilmente senso di esistere. Viene utilizzata durante il processo di esportazione per convertire un numero Double ovvero il valore DWORD (visto che in VB non è possibile definire Long senza segno) in una stringa esadecimale. L'istruzione *Hex* arriva infatti soltanto al 31° bit ed al superamento di questo, genera *Overflow*.

La funzione scompone il numero Double in due parti: la parte alta è mantenuta nella variabile **HighPart** mentre la parte bassa rimane nel parametro Numero. I due valori sono poi convertiti separatamente e le due stringhe unite solo in seguito alla conversione.

Il parametro Padding è utile per assicurare una lunghezza minima di quella specificata in tale argomento; eventuali cifre mancanti saranno riempite con zeri alla sinistra.

```
306.  Private Function Replace(ByVal Stringa As String, ByVal Trova As String, ByVal
      Rimpiazza As String) As String
307.    Dim Pos As Integer
308.    Dim Buffer As String
309.    Pos = InStr(1, Stringa, Trova)
310.    Buffer = ""
311.    Do While Pos > 0
312.      Buffer = Buffer & Left$(Stringa, Pos - 1) & Rimpiazza
313.      Stringa = Mid$(Stringa, Pos + Len(Trova))
314.      Pos = InStr(1, Stringa, Trova)
315.    Loop
316.    Replace = Buffer & Stringa
317.  End Function
318.
```

La funzione **Replace** è del tutto inutile ai programmatori VB6 ma per chi usufruisce (*come me*) della versione 5 di VB essa è indispensabile. Com'è ovvio, consente di sostuire tutte le occorrenze di una data stringa con un'altra e presenta alcune marginali differenze rispetto alla funzione *Replace* nativa in VB6.

Segue parte 4 >>

Fibia FBI
1 Aprile 2002
Corretto il 20 Settembre 2002

Torna all'indice degli HowTo

Utilizzare il registro di Windows (quarta parte)

http://www.vbsimple.net/howto/ht_044_4.htm

Difficoltà: 5 / 5

[<< Continua dalla parte 3]

Ci avviciniamo alla fine di questo lungo tutorial. Saranno trattate le ultime quattro funzioni della nostra classe *FBIRegistry* relative ai processi di esportazione ed importazione dei dati del registro in file.

```
319. Public Function Esporta(ByVal ChiavePrincipale As ChiaviPrincipali, ByVal
     SottoChiave As String, ByVal NomeFile As String, ByVal SottoChiavi As Boolean,
     ByVal Aggiunge As Boolean) As Boolean
320.   Dim FileNR As Integer
321.   Dim oldKey As Long
322.   Dim NomeChiave As String
323.
324.   If ChiavePrincipale = CHIAVE_APERTA Then ChiavePrincipale = lngKeyValue
325.   Select Case ChiavePrincipale
326.     Case HKEY_CLASSES_ROOT: NomeChiave = "HKEY_CLASSES_ROOT"
327.     Case HKEY_CURRENT_CONFIG: NomeChiave = "HKEY_CURRENT_CONFIG"
328.     Case HKEY_CURRENT_USER: NomeChiave = "HKEY_CURRENT_USER"
329.     Case HKEY_DYN_DATA: NomeChiave = "HKEY_DYN_DATA"
330.     Case HKEY_LOCAL_MACHINE: NomeChiave = "HKEY_LOCAL_MACHINE"
331.     Case HKEY_PERF_ROOT: NomeChiave = "HKEY_PERF_ROOT"
332.     Case HKEY_PERFORMANCE_DATA: NomeChiave = "HKEY_PERFORMANCE_DATA"
333.     Case HKEY_USERS: NomeChiave = "HKEY_USERS"
334.     Case Else
335.       MsgBox "Per salvare una chiave è necessario che la chiave principale sia una
     chiave di sistema.", vbCritical + vbOKOnly, "FBIRegistry"
336.       Esporta = False
337.       Exit Function
338.   End Select
```

La funzione **Esporta** consente il salvataggio di una chiave, delle sue sottochiavi e dei loro valori in un file di testo compatibile con Regedit. La funzione richiede tre argomenti fondamentali: la chiave di sistema in cui si trova la sottochiave, il percorso logico della sottochiave rispetto alla chiave di sistema ed il nome del file in cui salvare tali dati.

Gli altri due argomenti consentono di salvare soltanto la chiave specificata o anche tutte le sue sottochiavi. L'ultimo argomento specifica se creare un nuovo file di testo oppure aggiungere i dati a quelli già esistenti nel file.

Il controllo alle righe 325-338 assicura che la chiave specificata sia una chiave di sistema nell'enumerazione **ChiavePrincipali** ed in tal caso ne registra il nome nella variabile **NomeChiave**. Se la chiave specificata non è una chiave di sistema verrà generato un avviso e l'esportazione verrà interrotta.

```
339.   FileNR = FreeFile
340.   If Aggiunge Then
341.     Open NomeFile For Append As FileNR
```



```
342.    Else
343.      Open NomeFile For Output As FileNR
344.    End If
345.    If LOF(FileNR) = 0 Then Print #FileNR, "REGEDIT4" & vbNewLine
346.    oldKey = lngKeyValue
347.    lngKeyValue = 0
348.    Call RegOpenKeyEx(ChiavePrincipale, SottoChiave, ByVal 0&, lngKeySecurity,
     lngKeyValue)
349.    If Left$(SottoChiave, 1) <> "\" Then SottoChiave = "\" & SottoChiave
350.    If SottoChiave = "\" Then SottoChiave = ""
351.    SalvaRamo CHIAVE_APERTA, NomeChiave & SottoChiave, FileNR, SottoChiavi
352.    Print #FileNR, vbNullString
353.    Close FileNR
354.    FileNR = 0
355.    Call RegCloseKey(lngKeyValue)
356.    lngKeyValue = oldKey
357.    Esporta = True
358.  End Function
359.
```

In funzione dell'argomento ***Aggiungi*** verrà creato un nuovo file di testo sovrascrivendo i dati precedenti oppure il file sarà aperto in modalità di *Append* per inserire nuovi dati (righe 340-344). Inoltre se la dimensione del file aperto è di zero bytes (ovvero il file è nuovo oppure vengono aggiunti dati su un file vuoto) verrà inserita una breve intestazione per rendere il file riconoscibile da Regedit (riga 345).

Solo in seguito a questi necessari controlli verrà aperta la chiave richiesta per l'esportazione e passata con altri dati alla funzione **SalvaRamo** che vedremo tra poco. In sostanza la funzione SalvaRamo effettua l'operazione di esportazione dei dati vera e propria utilizzando l'handle di file passatole come argomento della funzione.

La nostra funzione **Esporta** infatti si occuperà soltanto dei controlli iniziali, dell'apertura del file, del richiamo della funzione **SalvaRamo** e della chiusura di file e chiave. Questo per dare completa flessibilità alla funzione **SalvaRamo** che non dovrà occuparsi di tutti questi noiosi compiti.

```
360.  Private Sub SalvaRamo(ByVal Chiave As ChiaviPrincipali, ByVal Percorso As String,
     ByVal FileNR As Integer, ByVal SottoChiavi As Boolean)
361.    Dim Conta As Integer
362.    Dim Conta2 As Long
363.    Dim Elementi As Variant
364.    Dim ValoreElemento As Variant
365.    Dim TipoDati As TipoValoriRegistro
366.    Dim Buffer As String
367.    Dim Buffer2 As String
368.    If Chiave = CHIAVE_APERTA Then Chiave = lngKeyValue
369.    Print #FileNR, "[" & Percorso & "]"
370.    Elementi = Me.ElencaValori(CHIAVE_APERTA)
371.    If Not IsNull(Elementi) Then
372.      For Conta = LBound(Elementi) To UBound(Elementi)
373.        ValoreElemento = Me.Valore(Elementi(Conta), TipoDati)
374.        Buffer2 = ""
375.        Elementi(Conta) = Replace(Elementi(Conta), "\", "\\")
376.        Elementi(Conta) = Replace(Elementi(Conta), vbCr, "\r")
377.        Elementi(Conta) = Replace(Elementi(Conta), vbLf, "\n")
378.        Elementi(Conta) = Replace(Elementi(Conta), """", "\""")
379.        Buffer = """" & Elementi(Conta) & """"
380.        If Elementi(Conta) = "" Then Buffer = "@"
```

La funzione SalvaRamo, la più complessa dell'intero progetto, effettua il salvataggio del contenuto di una chiave su un file già aperto, mantenendo naturalmente il formato dei dati originali. La funzione richiede quattro parametri: ***Chiave*** è la chiave che si intende salvare,

Percorso è il percorso logico della chiave nella struttura ad albero, ***FileNR*** è l'handle del file già aperto in cui scrivere i dati ed infine ***Sottochiavi*** determina se effettuare il salvataggio anche delle sottochiavi e dei valori in esse contenuti.

La funzione si apre con il salvataggio su file del percoso (riga 369) ed il recupero di tutti i valori presenti nella chiave specificata (riga 370). Per ogni valore recuperato sarà ottenuto anche il suo contenuto (riga 373) ed effettuate alcune semplici conversioni per rendere l'esportazione coerente con il formato REGEDIT4 (righe 375-380). Il valore predefinito della chiave deve essere salvato come `"@"`.

```
381.        Select Case TipoDati
382.          Case REG_SZ
383.            ValoreElemento = Replace(ValoreElemento, "\", "\\")
384.            ValoreElemento = Replace(ValoreElemento, vbCr, "\r")
385.            ValoreElemento = Replace(ValoreElemento, vbLf, "\n")
386.            ValoreElemento = Replace(ValoreElemento, """", "\""")
387.            Buffer2 = Buffer & "=""" & ValoreElemento & """"
```

Per fortuna o purtroppo il registro contiene una varietà di tipologie di dati presenti e soprattutto non tutti questi dati sono scritti nella maniera corretta. Per ogni tipo di dato trattato sarà necessario effettuare alcuni aggiustamenti in esportazione.

L'esempio più semplice è il formato stringa **REG_SZ** che richiede soltanto la sostituzione di alcuni caratteri speciali ("\", Enter, a capo, virgolette) nelle corrispondenti sequenze escape e ciò viene effettuato utilizzando la funzione **Replace**.

Purtroppo il caso degli altri tipi di dati non è altrettanto semplice. Esistono infatti alcuni dati che, pur essendo ad esempio valori **REG_DWORD** non contengono un valore corretto ma soltanto una matrice di bytes, quasi che si trattasse di dati **REG_BINARY**.

```
388.          Case REG_BINARY, REG_NONE, REG_EXPAND_SZ, _
389.               REG_DWORD_LITTLE_ENDIAN, REG_DWORD, _
390.               REG_MULTI_SZ
391.            If IsArray(ValoreElemento) Then
392.              Buffer = Buffer & "=hex"
393.              If TipoDati = REG_NONE Then Buffer = Buffer & "(0)"
394.              If TipoDati = REG_EXPAND_SZ Then Buffer = Buffer & "(2)"
395.              If TipoDati = REG_DWORD Then Buffer = Buffer & "(4)"
396.              If TipoDati = REG_MULTI_SZ Then Buffer = Buffer & "(7)"
397.              Buffer = Buffer & ":"
398.              If Not IsNull(ValoreElemento) Then
399.                For Conta2 = LBound(ValoreElemento) To UBound(ValoreElemento)
400.                  Buffer = Buffer & Right$("00" & LCase$(Hex$(ValoreElemento
     (Conta2))), 2)
401.                  If Conta2 < UBound(ValoreElemento) Then Buffer = Buffer & ","
402.                  If (Len(Buffer) > 76) And (Conta2 < UBound(ValoreElemento)) Then
403.                    Buffer2 = Buffer2 & Buffer & "\" & vbNewLine
404.                    Buffer = " "
405.                  End If
406.                Next Conta2
407.              End If
408.              Buffer2 = Buffer2 & Buffer
```

Nel caso del tipo di dati binario o DWORD dovrà quindi esser fatta un'investigazione più profonda. Il primo di questi controlli consiste nel verificare se i dati sono in forma di array ed in tal caso dovranno essere trattati come dati binari, pur manentendo il formato di dati originale. Tutti i dati binari devono essere salvati nel formato `"hex(x): XX, YY"`. dove x indica il tipo di dati e XX, YY sono i dati in forma binaria esadecimale. L'unica eccezione

sta nel caso dei dati **REG_BINARY**, che non richiedono le parentesi che racchiudono il tipo di dati. Tale formattazione di dati è fatta alle righe 392-397.

Soltanto adesso sarà possibile convertire i singoli bytes della matrice in stringhe esadecimali utilizzando l'istruzione *Hex$*. Ma, ahinoi, non ancora finito qui: ogni riga da salvare che superi la lunghezza di 76 caratteri dovrà essere spezzata in altre righe (righe 402-405).

```
409.           ElseIf TipoDati = REG_DWORD Then
410.             Buffer = Buffer & "=dword:"
411.             Buffer2 = Buffer & LCase$(HexDouble(ValoreElemento, 8))
412.           Else
413.             Buffer = Buffer & "=hex"
414.             If TipoDati <> REG_BINARY Then Buffer = Buffer & "(" & CStr(TipoDati) &
     ")"
415.             Buffer = Buffer & ":"
416.             If Not IsNull(ValoreElemento) Then Buffer = Buffer & CStr
     (ValoreElemento)
417.             Buffer2 = Buffer
418.           End If
419.         Case REG_DWORD_BIG_ENDIAN, REG_LINK, _
420.              REG_RESOURCE_LIST
421.           MsgBox "Tipo di dati non implementato!", vbCritical + vbOKOnly,
     "FBIRegistry"
422.       End Select
423.       Print #FileNR, Buffer2
424.     Next Conta
425.   End If
```

Se invece i dati recuperati non sono in forma di matrice, sarà verificato che il tipo di dati sia DWORD perché anche questo tipo richiede una sua particolare formattazione come `"dword: XX"` dove XX è il valore DWORD convertito in stringa esadecimale (righe 409-411).

Se i dati non sono né una matrice di bytes né un valore DWORD allora saranno semplicemente scritti come numero decimale (se i dati esistono) oppure non verranno scritti affatto. Sarà cioè scritto soltanto il tipo di dati trattato come `"hex(x):"` ma non il loro contenuto. Non si tratta infatti di una possibilità remota ma di una consuetudine nell'universo del registro di Windows.

I tipi di dati **REG_DWORD_BIG_ENDIAN**, **REG_LINK** e **REG_RESOURCE_LIST** non saranno affatto trattati e quindi esclusi dal processo di esportazione.

Alla riga 423 verrà infine scritto il buffer di dati preparato mediante i controlli e le conversioni appena fatti.

```
426.   If (SottoChiavi = True) And (Me.NumeroSottoChiavi > 0) Then
427.     Elementi = ElencaChiavi(CHIAVE_APERTA)
428.     If IsNull(Elementi) Then Exit Sub
429.     For Conta = LBound(Elementi) To UBound(Elementi)
430.       Print #FileNR, vbNullString
431.       Conta2 = lngKeyValue
432.       Call RegOpenKeyEx(Chiave, Elementi(Conta), ByVal 0&, lngKeySecurity,
     lngKeyValue)
433.       SalvaRamo CHIAVE_APERTA, Percorso & "\" & Elementi(Conta), FileNR,
     SottoChiavi
434.       Call RegCloseKey(lngKeyValue)
435.       lngKeyValue = Conta2
436.     Next Conta
437.   End If
```



```
438. End Sub
439.
```

Un'intera chiave è stata salvata su file. Resta pertanto l'ultima possibilità ovvero quella di dover salvare ogni singola sottochiave della nostra chiave e così le sottochiavi delle sottochiavi, etc... Quest'operazione è svolta nel ciclo descritto alle righe 429-436.

Sarà quindi aperta una sottochiave per volta, ricreato il percorso e richiamata ricorsivamente la funzione **SalvaRamo** con i nuovi dati. La ricorsività assicurerà il salvataggio di tutti i valori presenti in ogni sottochiave di ogni sottochiave della prima chiave.

Questa è la ragione principale per cui abbiamo preferito separare la routine **Esporta** da quella **SalvaRamo**.

Segue parte 5 >>

Fibia FBI
1 Aprile 2002
Corretto il 20 Settembre 2002

Torna all'indice degli HowTo

