



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](#)
