



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 = CDb1(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 &H7FFFFFF0) \ 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](#)
