Estrarre i singoli bytes da un dato di tipo multibyte

http://www.vbsimple.net/howto/ht_031.htm

Difficoltà: 3 / 5

In Visual Basic, come in altri linguaggi, esistono determinati tipi di dati semplici che, mettendo assieme più bytes, formano un unico valore quale ad esempio i numeri Integer o i Long.
Il tipo di dati più semplice è il byte ed ogni variabile o costante di tale tipologia occupa sempre e soltanto un byte, ovvero un carattere o che si dica, una singola cella di memoria. Poiché un byte si compone di 8 bit, il massimo numero che tale unità può contenere equivale a 2^8, quindi 255 (poiché è necessario includere anche il numero 0).

Tutti gli altri tipi di dati si compongono di due o più bytes concatenati tra loro. Così, ad esempio il tipo Integer (che permette il conteggio fino a 2^16, ovvero da -32.768 a +32.767) si compone di due bytes, uniti tra loro in una singola entità.
Alla stessa maniera i dati di tipo Long sono formati da 4 bytes, ovvero 32 bit, e permettono la memorizzazione di numeri compresi tra -2.147.483.648 e +2.147.483.647 (2^32).

In alcuni casi di programmazione a basso livello come ad esempio la manipolazione di immagini, può essere necessario estrarre i singoli bytes di cui si compone un dato di tipo multibyte, ovvero composto da più bytes.

I bytes di un valore sono consecutivi tra loro, così dato l'indirizzo inziale di un dato di un certo tipo, i bytes di cui si compone sono consecutivi tra loro. Tuttavia, nelle architetture **Intel x86** i dati non sono memorizzati in maniera continua (**Big-Endian**) ma l'ordine dei bytes è invertito; così il primo byte di un tipo multibyte si trova all'ultima posizione. Questo processo di memorizzazione invertita è detto **Little-Endian**.
Per ulteriori informazioni su questo tipo di memorizzazione vedi le informazioni aggiuntive.

Per effettuare tale separazione di bytes o unione in un tipo multibyte è necessario sfruttare la funzione API ***CopyMemory***, in realtà alias di un'altra funzione: ***RtlMoveMemory***. Mediante tale funzione leggeremo i dati dall'indirizzo iniziale del valore multibyte e li scriveremo da un'altra parte.
Questo esempio effettua sia la lettura dei bytes contenuti in un dato multibyte sia la costruzione di un numero multibyte partendo da un array di bytes.

Il nostro semplice form si compone di pochi controlli: innanzitutto sulla prima riga abbiamo una *Label* di **LongHEXLabel** con la proprietà *Caption* impostata a ***"Numero Long Esadecimale"***. A fianco d'essa abbiamo la corrispondente *TextBox* di nome **LongHEX**, con la proprietà *MaxLength* impostata ad 8 (permettendo quindi l'inserimento fino a 8 cifre esadecimali). Segue a questa un'altra *Label* di nome **LongDECLabel**, con

la proprietà *AutoSize* impostata a True, che verrà utilizzata per mostrare il risultato decimale del numero esadecimale inserito.

La seconda riga di controlli contiene un'altra *Label* di nome **HEXArrayLabel** con il solo scopo descrittivo. Segue a questa una matrice di 4 *TextBox* di nome **HEXArray** con proprietà *Index* da 0 a 3, tutte con proprietà *MaxLength* impostata a 2.

Completiamo il form con due *CommandButton* di nome **PulsanteLongBytes** e **PulsanteBytesLong** e *Caption* impostate a ***"Long -> Bytes"*** e ***"Bytes -> Long"***.

Il funzionamento del form è semplice. L'utente immette un numero esadecimale nella casella di testo più grande, preme il pulsante ***"Long -> Bytes"*** ed ottiene i singoli bytes del numero inserito, ovviamente però in ordine inverso per i motivi accennati in precedenza. Altresì l'utente potrà immettere 4 numeri esadecimali nelle 4 caselle di testo più piccole, premere il pulsante ***"Bytes -> Long"*** ed ottenere così un corrispondente numero multibyte. Il tipo multibyte utilizzato sarà il Long che contiene al suo interno 4 bytes.

Definiamo subito il codice delle due funzioni di conversione:

```
1. Option Explicit
2. Private Declare Sub CopyMemory Lib "KERNEL32" Alias "RtlMoveMemory" (ByRef
   Destination As Any, ByRef Source As Any, ByVal numbytes As Long)
3. Private CARATTERIHEX As String
4.
```

Alla riga 2 abbiamo innanzitutto definito la funzione API **CopyMemory**, alias della vera funzione, **RtlMoveMemory**. Essa richiede tre parametri: due d'essi sono puntatori mentre il terzo è un numero semplice. Il primo parametro (*Destination*) è il puntatore all'indirizzo di memoria di destinazione, in cui verranno scritti i dati; il secondo parametro (*Source*) è un puntatore all'area di memoria da cui leggere i dati; il terzo parametro (*numbytes*) definisce il numero di bytes da copiare dall'indirizzo Source all'indirizzo Destination.

Alla riga 3 abbiamo definito una variabile di nome **CARATTERIHEX** che utilizzeremo più avanti per controllare che i dati immessi dall'utente siano effettivamente validi come numero esadecimale.

```
5. Public Sub LongToBytes(ByRef TheArray() As Byte, ByRef TheLong As Long)
6.     Call CopyMemory(TheArray(LBound(TheArray)), TheLong, 4)
7. End Sub
8.
```

La prima funzione che vedremo è la ***LongToBytes*** che, estrae i singoli bytes dal numero Long e li scrive in un array di bytes.
La funzione è semplicissima: in chiamata riceve due puntatori e li passa entrambi alla funzione API CopyMemory. Ad essi aggiunge il terzo parametro (4) per definire l'ampiezza del dato trattato, 4 bytes.

Possiamo notare una strana scrittura: **`TheArray(LBound(TheArray))`**
Ciò serve per determinare quale sia il primo effettivo elemento dell'array di nome TheArray.
In situazioni normali il primo elemento è 0, ma in casi particolari (mediante l'utilizzo

dell'istruzione *Option Base 1*) tale limite può essere modificato. Così facendo la funzione automaticamente calcolerà l'indice del primo elemento della matrice.

Tale matrice sarà il contenitore per i 4 bytes letti a partire dall'indirizzo del puntatore **TheLong**.

```
 9. Public Function BytesToLong(ByRef TheArray() As Byte) As Long
10.     Dim TempLong As Long
11.     Call CopyMemory(TempLong, TheArray(LBound(TheArray)), 4)
12.     BytesToLong = TempLong
13. End Function
14.
```

La seconda funzione effettuerà l'operazione inversa della prima, ovvero leggerà i dati da un array di bytes ed otterrà con essi un numero Long che riporterà come risultato della funzione.

La chiamata alla funzione CopyMemory è quasi identica alla precedente. Ciò che cambia è l'ordine dei parametri; infatti questa volta la destinazione sarà una variabile temporanea di nome **TempLong** e l'origine sarà la matrice di bytes passata alla funzione.
Tale chiamata leggerà i dati della matrice e li scriverà nella variabile TempLong. La funzione si chiude riportando in uscita il numero TempLong (riga 12).

Seguono quattro routines dedicate esclusivamente al controllo dei dati inseriti dall'utente. Ciò per evitare di passare alle funzioni API devi valori errati.

```
15. Private Sub Form_Load()
16.     CARATTERIHEX = "0123456789ABCDEF"
17.     CARATTERIHEX = CARATTERIHEX & Chr(vbKeyBack)
18. End Sub
19.
```

Durante il caricamento del form inizializziamo la variabile **CARATTERIHEX** con i sedici simboli del sistema numerico esadecimale. Ad essa aggiungiamo il simbolo ASCII del tasto *BackSpace*. Infatti la variabile **CARATTERIHEX** servirà per controllare se l'input inserito dall'utente è valido. Se non avessimo inserito il simbolo del *BackSpace* l'utente non avrebbe potuto cancellare i numeri precedenti se non utilizzando il tasto *Canc*.

Sarebbe stato più conveniente dichiarare la variabile CARATTERIHEX come costante ma non è stato possibile a causa della funzione *Chr* da concatenare.

```
20. Private Sub LongHEX_Change()
21.     LongDECLabel.Caption = CLng(Format("&H0" & LongHEX.Text))
22. End Sub
23.
```

Al cambiamento del contenuto della casella di testo **LongHEX** il valore nella Label **LongDECLabel** viene aggiornato calcolando il valore decimale del numero esadecimale inserito.

La funzione di conversione da esadecimale è la notissima *Format* con l'appoggio della stringa ***"&H0"*** che assicura che il numero non sia nullo e richiede la conversione da esadecimale.

```
24. Private Sub HEXArray_KeyPress(Index As Integer, KeyAscii As Integer)
```



```
25.     KeyAscii = Asc(UCase(Chr(KeyAscii)))
26.     If InStr(1, CARATTERIHEX, Chr(KeyAscii)) = 0 Then KeyAscii = 0
27. End Sub
28.
29. Private Sub LongHEX_KeyPress(KeyAscii As Integer)
30.     KeyAscii = Asc(UCase(Chr(KeyAscii)))
31.     If InStr(1, CARATTERIHEX, Chr(KeyAscii)) = 0 Then KeyAscii = 0
32. End Sub
33.
```

Alla pressione di un tasto nelle varie caselle di testo deve essere controllato se il tasto premuto è effettivamente una cifra del sistema esadecimale.

Pertanto, alle righe 25 e 30 viene portata l'eventuale lettera in maiuscolo (non è obbligatorio ma migliora l'estetica del programma) ed in seguito viene controllato se tale valore compare all'interno della stringa **CARATTERIHEX** che conterrà tutti i simboli esadecimali. Se il simbolo compare, l'istruzione *Instr* restituirà la posizione in cui tale carattere compare, quindi un numero maggiore di 0.
Nel caso che il simbolo non venga trovato all'interno della stringa l'istruzione *Instr* restituirà il valore 0. In questo caso (simbolo non trovato, quindi non valido) il tasto premuto sarà invalidato impostando il parametro ***KeyAscii*** a 0.

Segue il codice dei due pulsanti di estrazione ed unione.

```
34. Private Sub PulsanteLongBytes_Click()
35.     Dim NUMERI(3) As Byte
36.     LongToBytes NUMERI, CLng(Format("&H0" & LongHEX.Text))
37.     HEXArray(0).Text = Hex(NUMERI(0))
38.     HEXArray(1).Text = Hex(NUMERI(1))
39.     HEXArray(2).Text = Hex(NUMERI(2))
40.     HEXArray(3).Text = Hex(NUMERI(3))
41. End Sub
42.
```

La pressione del pulsante **PulsanteLongBytes** richiede l'estrazione dei singoli bytes contenuti nel numero specificato nella casella di testo *LongHEX*.

Pertanto dichiariamo una nuova matrice di tipo Byte (riga 35), richiamiamo la funzione ***LongToBytes*** che si occuperà di riempirla, passandole come parametri la matrice ed il numero, convertito a Long decimale, contenuto nella casella **LongHEX** (riga 36).

Fatto questo potremo andare a leggere i valori dalla matrice e scriverli nella matrice di caselle di testo (righe 37-40).

```
43. Private Sub PulsanteBytesLong_Click()
44.     Dim NUMERI(3) As Byte
45.     NUMERI(0) = CByte(Format("&H0" & HEXArray(0).Text))
46.     NUMERI(1) = CByte(Format("&H0" & HEXArray(1).Text))
47.     NUMERI(2) = CByte(Format("&H0" & HEXArray(2).Text))
48.     NUMERI(3) = CByte(Format("&H0" & HEXArray(3).Text))
49.     LongHEX.Text = Hex(BytesToLong(NUMERI))
50. End Sub
```

L'operazione inversa è gestita dal click sul pulsante **PulsanteBytesLong**.
Come prima dichiariamo una nuova matrice di Bytes e la riempiamo con i valori, convertiti in decimale delle 4 caselle di testo (righe 44-48).
Riempita la matrice potremo richiamare la funzione ***BytesToLong*** passandole la semplice

matrice. In uscita la funzione ci riporterà il numero Long generato. Passeremo tale numero, convertito in esadecimale mediante la funzione *Hex*, all'interno della casella **LongHEX**.

Il nostro esempio è completo. Il progetto leggerà i dati dalle caselle di testo e li convertirà da un formato all'altro.

L'esecuzione del progetto mostrerà molto chiaramente la strana struttura Little-Endian dei dati nelle macchine **Intel x86**.

Naturalmente esistevano tantissime altre soluzioni per effettuare quest'operazione ma questa è quella più veloce e flessibile ai possibili cambiamenti.

Fibia FBI
Tratto dalla MS Knowledge Base ID Q171652
28 Marzo 2001

Torna all'indice degli HowTo

