



## Gestire i colori RGB

[http://www.vbsimple.net/howto/ht\\_038.htm](http://www.vbsimple.net/howto/ht_038.htm)

Difficoltà: 2 / 5

Nella maggioranza dei programmi per personal computer i colori sono gestiti come triplette **RGB** (Red Green Blue - Rosso Verde Blu). Anche Visual Basic adotta tale tecnica; mettendo insieme 3 bytes ( $3 * 8 = 24$  bit) possiamo ottenere 16.777.216 ( $2^{24}$ ) differenti colori. Questo mettere insieme delle diverse gradazioni di ogni componente del colore deve essere fatto secondo un certo criterio.

Vedremo in questo HowTo come maneggiare i colori, comporli in numeri complessi e scomporli in triplette RGB attraverso due diverse soluzioni.

Utilizzeremo un solo form con vari controlli; cominciamo dalla parte superiore in cui una serie di controlli sono contenuti in un frame: abbiamo innanzitutto tre **Label A** di nome **LabelColori** in una **matrice** di 3 elementi con il nome dei 3 colori primari.



Accanto ad ognuna di esse c'è una **barra di scorrimento orizzontale** di nome **BarraColori** con gli indici da 0 a 2, proprietà **Min** impostata a 0, proprietà **Max** impostata a 255, proprietà **SmallChange** impostata su 1 e **LargeChange** impostata su 8. Questo perché ogni componente dei colori utilizza 1 **byte** e pertanto può contenere i numeri compresi tra 0 e 255.

Accanto ad ogni barra è presente una **TextBox** di nome **TextColoriDec** con indici sempre da 0 a 2, proprietà **MaxLength** impostata a 3 e **Locked** impostata a **True**. Esse mostreranno soltanto il valore corrente di ogni singola barra del colore. Infine altre 3 **TextBox** di nome **TextColoriHex** con proprietà **MaxLength** impostata a 2 e **Locked** impostata a **True**. Queste ultime serviranno a rappresentare il valore esadecimale del colore indicato nella casella a fianco e quindi dalle barre di scorrimento.


La parte inferiore del form contiene invece i comandi utilizzabili nel nostro programma. Il suo ruolo è:

- Mettere assieme una tripletta di componenti RGB in un numero lungo di colore;
- Scomporre un numero lungo di colore in una tripletta di colori RGB.

Per entrambi i problemi abbiamo due soluzioni: una semplice e banale ed una tecnicamente più corretta che svolge le operazioni più elementari per ottenere il risultato.

Ritornando al nostro form, abbiamo una **TextBox** di nome **TextColoreLungo** con la proprietà **MaxLength** impostata a 8; sulla destra invece tre caselle di testo di nome **ColoriRGB** con indici da 0 a 2 e proprietà **MaxLength** impostata a 3.

In mezzo alle caselle di testo 4 CommandButton per eseguire le quattro operazioni previste di nome **Long2RGB**, **Long2RGB2**, **RGB2Long** e **RGB2Long2**. I primi due convertiranno il numero lungo in tripletta RGB mentre gli altri due congiungeranno la tripletta RGB in un numero lungo di colore.


Infine sulla destra del form ci sono due *PictureBox*  di nome **AnteprimaColore1** e **AnteprimaColore2**; la prima servirà per vedere il colore indicato dalle barre di scorrimento della parte superiore del form mentre la seconda servirà per mostrare il colore indicato dalle 3 caselle di testo in basso.

Vediamo il codice che, preso passo passo, non presenta alcuna difficoltà particolare, ma molto utile didatticamente per capire la gestione dei colori in Visual Basic.

```

1. Option Explicit
2.
3. Private Sub BarraColori_Scroll(Index As Integer)
4.     TextColoriDec(Index).Text = CStr(BarraColori(Index).Value)
5.     TextColoriHex(Index).Text = Right$("00" & Hex$(BarraColori(Index).Value), 2)
6.     AnteprimaColore1.BackColor = RGB(BarraColori(0).Value, BarraColori(1).Value,
    BarraColori(2).Value)
7. End Sub
8.
9. Private Sub BarraColori_Change(Index As Integer)
10.    BarraColori_Scroll Index
11. End Sub
12.

```

Nel momento in cui l'utente "*afferra*" la barra di scorrimento e la posta da un valore ad un altro viene scatenato l'[evento](#)  **Scroll**. Essendo una matrice di controlli viene passato il parametro Index in base alla barra che si sta trascinando.

Alla riga 4 viene aggiornata la casella di testo di nome **TextColoriDec** corrispondente alla barra trascinata con il valore della barra. Alla riga successiva viene svolta l'operazione analoga ma in esadecimale aggiornando la casella di testo **TextColoriHex**. L'operazione di conversione viene svolta dalla funzione *Hex\$*. L'istruzione *Right\$* serve per assicurare che vengano mostrati almeno 2 caratteri [esadecimali](#). Alla riga 6 viene infine aggiornata la *PictureBox* **AnteprimaColore1** con il colore indicato da *tutti e tre* i valori delle barre di scorrimento.

Se l'utente invece di trascinare la barra di scorrimento clicca su un'area libera della barra oppure su una delle freccette della stessa oppure utilizza la tastiera, viene aggiornato il valore della barra ma non viene eseguito l'evento **Scroll**. Al suo posto è eseguito l'evento **Change**; ecco perché abbiamo fatto in modo che l'esecuzione di tale evento richiami (riga 10) la funzione legata all'evento **Scroll**, in modo da non dover ripetere lo stesso codice.

```

13. Private Sub ColoriRGB_GotFocus(Index As Integer)
14.    ColoriRGB(Index).SelStart = 0
15.    ColoriRGB(Index).SelLength = 3
16. End Sub
17.
18. Private Sub ColoriRGB_KeyPress(Index As Integer, KeyAscii As Integer)
19.    If ((KeyAscii < Asc("0")) Or (KeyAscii > Asc("9"))) And (KeyAscii <> vbKeyBack)
    Then KeyAscii = 0
20. End Sub
21.

```

La funzione alle righe 13-16 è esclusivamente di supporto e non ha nessuna utilità nel progetto. Nel momento in cui una TextBox di nome ColoriRGB riceve il focus il testo contenuto in essa viene selezionato, cosicché la digitazione sovrascrive i dati precedenti.

Dovendo tali caselle di testo contenere soltanto numeri abbiamo fatto in modo di bloccare tutti i tasti eccetti i numeri ed il BackSpace per cancellare. Alla riga 19 viene effettuato il controllo del tasto premuto. Possiamo scomporre la condizione in due parti:

```
If ((KeyAscii < Asc("0")) Or (KeyAscii > Asc("9")))
```


Se il tasto premuto è minore del codice del tasto 0 oppure maggiore del codice del tasto 9, ovvero non è un tasto compreso tra 0 e 9...

```
And (KeyAscii <> vbKeyBack) Then KeyAscii = 0
```

Ma altresì il tasto premuto **non** è il tasto BackSpace, allora tale tasto deve essere annullato e non mandato alla casella di testo. Questo duplice controllo assicura che il tasto premuto sia un numero oppure il tasto BackSpace.

Questo controllo farà sì che siano inseriti soltanto numeri nella casella di testo. L'unico problema può consistere nel copia-incolla di dati nella casella. Abbiamo voluto evitare questo ulteriore controllo per non creare confusione.

```
22. Private Sub ColoriRGB_Change(Index As Integer)
23.     If Val(ColoriRGB(Index).Text) > 255 Then
24.         ColoriRGB(Index).Text = "255"
25.         ColoriRGB(Index).SelStart = 0
26.         ColoriRGB(Index).SelLength = 3
27.     End If
28.     AnteprimaColore2.BackColor = RGB(Val(ColoriRGB(0).Text), Val(ColoriRGB
    (1).Text), Val(ColoriRGB(2).Text))
29. End Sub
30.
```

Controlliamo anche l'evento  **Change** delle caselle di testo **ColoriRGB** che verrà eseguito subito dopo l'evento KeyPress. Questo per evitare che il numero immesso in ciascuna casella sia inferiore a 256 e per aggiornare la PictureBox con l'anteprima del colore.

La prima di queste due operazioni viene svolta dalle righe 23-27; se il numero immesso è maggiore di 255 allora ne cambia il contenuto in 255 e ne evidenzia il testo in essa.

Alla riga 28 viene invece cambiato il colore di **AnteprimaColore2** con i valori delle tre caselle di testo **ColoriRGB**. Viene utilizzata la funzione RGB che combina i tre valori.

Seguono le due funzioni di conversione da numero lungo a tripletta RGB e viceversa. Prima vediamo però come sono organizzati i colori.

Data una tripletta di valori che rappresentano ogni colore, ad esempio i numeri **48**, **128** e **226** hanno i corrispondenti valori esadecimali **30**, **80** e **E2**. Sia in forma decimale che esadecimale tali valori occupano sempre due bytes. Il sistema per creare un numero lungo rappresentativo di una tripletta di colori mette assieme i tre singoli valori: **E28030**. Tale numero convertito in decimale è **14843952**. Tale colore rappresenta un **azzurro**.

Spiegato in altri termini, la conversione da tripletta a numero lungo viene effettuata

facendo la somma di tali valori: **E20000** + **008000** + **000030**. Il valore **E2** viene moltiplicato per **65536** (ovvero  $256^2$  oppure **10000** esadecimale); il valore **80** viene moltiplicato per **256** (ovvero  $256^1$  oppure **100** esadecimale); mentre il valore del blu **30** viene lasciato inalterato (ovvero  $256^0$  oppure **1** esadecimale).

Lo stesso sistema può essere effettuato con cifre decimali:

**226** \* **65536** = 14811136 (ovvero **E20000** esadecimale);

**128** \* **256** = 32768 (ovvero **008000** esadecimale);

**48** \* **1** = 48 (ovvero **000030** esadecimale).

La somma di  $14811136 + 32768 + 48$  dà come risultato **14843952** ovvero il nostro valore lungo corrispondente (**E28030** in esadecimale).

Pertanto la conversione opposta deve essere fatta mediante operazioni inverse: essendo i valori sommati tra loro è necessario estrarre ogni parte corrispondente mediante divisioni intere ed [operazioni booleane di AND](#) che assicurano che vengano estratti soltanto gli ultimi due bytes del numero

Ricordando che l'operatore `\` indica una divisione intera (ad esempio  $7 \setminus 3 = 2$ ), il numero lungo 14843952 può essere scomposto come segue:

$14843952 \setminus 1 \text{ AND } 255 = 48$  (ovvero **30** esadecimale)

$14843952 \setminus 256 \text{ AND } 255 = 128$  (ovvero **80** esadecimale)

$14843952 \setminus 65536 \text{ AND } 255 = 226$  (ovvero **E2** esadecimale)

Abbiamo due soluzioni per ogni problema.

```

31. Private Sub Long2RGB_Click()
32.     Dim Valore As String
33.     Valore = Right$("000000" & Hex$(TextColoreLungo.Text), 6)
34.     ColoriRGB(0).Text = CInt("&H" & Mid(Valore, 5, 2))
35.     ColoriRGB(1).Text = CInt("&H" & Mid(Valore, 3, 2))
36.     ColoriRGB(2).Text = CInt("&H" & Mid(Valore, 1, 2))
37. End Sub
38.
```

La prima soluzione effettua una semplice conversione del numero lungo in una stringa esadecimale ed estrae le coppie di byte corrispondenti per ogni colore.

Alla riga 33 viene effettuata la conversione del numero in esadecimale (funzione *Hex\$*); il numero di caratteri restituiti è sempre 6, in base all'utilizzo dell'istruzione *Right\$*. Le righe 34-36 estraggono 3 coppie di byte dal valore convertito. Il rosso è indicato dalle 2 cifre meno significative mentre il blu sta alla prima posizione.

```

39. Private Sub RGB2Long_Click()
40.     TextColoreLungo.Text = CStr(RGB(Val(ColoriRGB(0).Text), Val(ColoriRGB(1).Text),
        Val(ColoriRGB(2).Text)))
41. End Sub
42.
```

La conversione da tripletta RGB a numero lungo è ancora più semplice e viene effettuato utilizzando la funzione apposita *RGB*, convertita in stringa per visualizzarla nella casella di testo. La funzione *RGB* richiede che siano passati in ordine i colori rosso, verde e blu.

Seguono altre due soluzioni per produrre i medesimi risultati che effettuano le operazioni

matematiche dovute per ottenere il risultato. In pratica è quello che avviene *"under the hood"* utilizzando le soluzioni appena viste.

```

43. Private Sub Long2RGB2_Click()
44.     If TextColoreLungo.Text = "" Then TextColoreLungo.Text = "0"
45.     ColoriRGB(0).Text = CStr(CLng(TextColoreLungo.Text) \ 1 And 255)
46.     ColoriRGB(1).Text = CStr(CLng(TextColoreLungo.Text) \ 256 And 255)
47.     ColoriRGB(2).Text = CStr(CLng(TextColoreLungo.Text) \ 65536 And 255)
48. End Sub
49.

```

La conversione da numero lungo a tripletta RGB viene effettuata mediante le operazioni di divisione intera ed operazioni di AND, come visto prima.

Alla riga 44 viene verificato che il valore di **TextColoreLungo** non sia una stringa vuota; se lo fosse sarebbe sostituita con uno 0. Questo al fine di evitare errori di calcolo su numeri impossibili.

Le righe 45-48 estraggono i 3 bytes nell'ordine rosso, verde e blu ed inseriscono il risultato nelle caselle **ColoriRGB** corrispondenti.

```

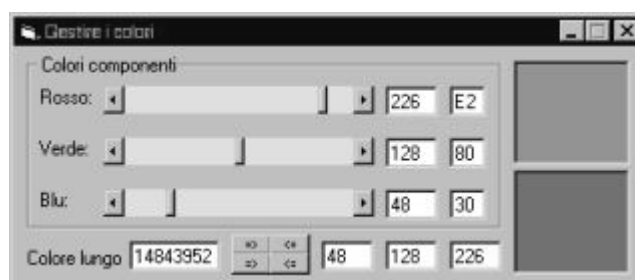
50. Private Sub RGB2Long2_Click()
51.     Dim Valore As Long
52.     If ColoriRGB(0).Text = "" Then ColoriRGB(0) = "0"
53.     If ColoriRGB(1).Text = "" Then ColoriRGB(1) = "0"
54.     If ColoriRGB(2).Text = "" Then ColoriRGB(2) = "0"
55.     Valore = 0
56.     Valore = Valore + CLng(ColoriRGB(0).Text) * 1
57.     Valore = Valore + CLng(ColoriRGB(1).Text) * 256
58.     Valore = Valore + CLng(ColoriRGB(2).Text) * 65536
59.     TextColoreLungo.Text = CStr(Valore)
60. End Sub

```

L'operazione di conversione da tripletta a valore lungo è ancora più semplice. Si tratta infatti di moltiplicare i valori di ogni colore rispettivamente per  $255^0$ ,  $255^1$  e  $255^2$  e sommarli tra loro.

I controlli alle righe 52-54 assicurano che sia presente un valore nelle caselle **ColoriRGB**. Alla riga 55 comincia la conversione: viene azzerata la variabile **Valore**; il valore del rosso viene moltiplicato per **1** ( $256^0$ ) e sommato a **Valore**. Alla riga successiva viene moltiplicato il valore del verde per **256** ( $256^1$ ) e sommato a **Valore**; il blu viene invece moltiplicato per **65536** ( $256^2$ ) ed il risultato viene sempre sommato a **Valore**. Infine la casella **TextColoreLungo** assumerà il contenuto di **Valore**.

Questo processo di conversione viene applicato dalla quasi totalità dei programmi. Alcuni fanno piccole eccezioni; ad esempio l'HTML, il linguaggio di marcatura con cui vengono scritte le pagine web come questa utilizza lo stesso procedimento ma inverte i colori rosso e blu.



Lo stesso colore esadecimale **E28030** che in Visual Basic rappresentava un **azzurro**, nell'HTML rappresenta invece un **marroncino**. Per ottenere lo stesso azzurro di Visual Basic è necessario invertire i colori rosso e blu, ottenendo quindi **3080E2**.

[Fibia FBI](#)

27 Agosto 2001



[Torna all'indice degli HowTo](#)

---