


Scrivere del testo sulla Console (StdOut)

http://www.vbsimple.net/howto/ht_037.htm

Difficoltà:  3 / 5

Prima dell'avvento di Windows 95 con la sua interfaccia grafica erano comunissimi i programmi che mostravano il loro output sulla [console](#), ovvero quella finestra che ora è chiamata Prompt di MS-DOS . La particolarità di tali programmi era la possibilità di poter ridirigere il testo di output su dispositivi (quali stampanti, porte seriali o altro) e su files mediante il segno di maggiore (>) seguito dal nome del dispositivo o file a cui inviare il testo.


Il canale che collega in output un programma con la console è detto Standard Output (*StdOut*), uno dei tanti [stream](#) presenti nel linguaggio C. Lo StdOut potrà essere reindirizzato su un file nella maniera classica.

I programmi con interfaccia grafica hanno quasi abolito questo genere di operazioni e non è più possibile redirigere l'output del programma su un file a meno che il programma non contempli una funzione apposita per farlo.

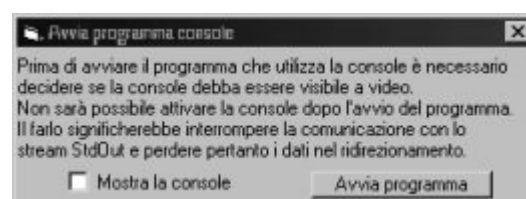
A prima vista questo genere di programmi sembrano obsoleti ma dedicando maggiore attenzione possiamo notare che in tutti i sistemi operativi moderni sono presenti ancora svariati programmi con interfaccia testuale. Questo avviene per permettere le comunicazioni tra diversi sistemi in una maniera sicura; contemporaneamente con il sempre più maggiore impiego di Internet, vanno sviluppandosi i programmi che quando richiamati inviano i loro risultati sul browser generando quindi pagine in maniera dinamica, in base alla richiesta. Questi ultimi programmi che si interfacciano con i browser sono detti [CGI](#) ([Common Gateway Interface](#)) ed anch'essi scrivono i loro risultati sullo StdOut che il browser reindirizzerà su se stesso per ricevere l'output.

Il nostro progetto concederà due possibilità: redirigere del testo sullo StdOut oppure aprire una finestra Console e scrivere il testo in quella. La seconda opzione è utile per provare il programma e quando il programma viene richiamato senza redirezionamento ma sarà inutilizzabile quando è in atto il redirezionamento dell'output.

Il progetto conterrà due form molto semplici: il primo servirà per decidere se aprire la finestra Console e lanciare il programma che si trova nel secondo form.

Il primo form si compone soltanto di una *CheckBox* ☒ di nome **MostraConsole** e di un *CommandButton*  di nome **Avvia**.

Nel nostro form abbiamo inserito anche una *Label* descrittiva ma la sua funzione è praticamente nulla. Il codice è molto semplice:




```

1. Option Explicit
2. Private Declare Function AllocConsole Lib "KERNEL32" () As Long
3. Private Declare Function SetConsoleTitle Lib "KERNEL32" Alias
   "SetConsoleTitleA" (ByVal lpConsoleTitle As String) As Long
4.
5. Private Sub Avvia_Click()
6.     If MostraConsole.Value = vbChecked Then
7.         Call AllocConsole
8.         Call SetConsoleTitle("Utilizzo della console all'interno di VB5")
9.     End If
10.    Load FormConsole
11.    FormConsole.Show vbModeless
12.    Unload Me
13. End Sub

```

Alla riga 2 abbiamo dichiarato la funzione [API](#) *AllocConsole* che apre una nuova finestra Console. Alla riga 3 invece abbiamo un'altra funzione API relativa alla Console; essa è *SetConsoleTitle* e verrà utilizzata per modificare il titolo presente sulla barra del titolo della Console.

L'unico [evento](#) ⚡ utilizzato sarà il Click sopra il pulsante **Avvia**. Sarà controllato lo stato della CheckBox **MostraConsole**. Se essa è selezionata saranno richiamate le due funzioni API dichiarate prima, per aprire la Console ed impostare il titolo della sua finestra. Fatto questo, sarà caricato e mostrato il secondo Form mentre il primo si avvia a chiudersi.

Il secondo form è persino più semplice del primo: contiene, infatti, soltanto una TextBox di nome **Testo** con la proprietà  *Multiline* impostata a True. La casella di testo riceverà i dati che andranno scritti sulla Console o sullo StdOut.



Il codice invece è un po' più complesso:

```

1. Option Explicit
2. Private Declare Function FreeConsole Lib "KERNEL32" () As Long
3. Private Declare Function GetStdHandle Lib "KERNEL32" (ByVal nStdHandle As Long) As
   Long
4. Private Declare Function CloseHandle Lib "KERNEL32" (ByVal hObject As Long) As Long
5. Private Declare Function WriteFile Lib "KERNEL32" (ByVal hFile As Long, ByVal
   lpBuffer As String, ByVal nNumberOfBytesToWrite As Long, lpNumberOfBytesWritten As
   Long, lpOverlapped As Any) As Long
6. Private Const STD_INPUT_HANDLE = -10&
7. Private Const STD_OUTPUT_HANDLE = -11&
8. Private Const STD_ERROR_HANDLE = -12&
9.
10. Private hConsole As Long
11.

```

Alla riga 2 viene dichiarata la funzione API *FreeConsole* che servirà per chiudere la finestra Console eventualmente aperta dal primo form.

Alla riga 3 è dichiarata la funzione API *GetStdHandle* che ottiene un [handle](#) specifico. Nel nostro caso verrà utilizzata per ottenere l'handle dello StdOut. Lo stesso handle andrà poi chiuso al termine dell'utilizzo del programma mediante la funzione API *CloseHandle* (trattata anche in un [altro HowTo](#)) dichiarata alla riga 4.

L'ultima funzione API è *WriteFile* (riga 5) che, come indica il nome, serve per scrivere dei dati in un file. La funzione richiede l'handle del file su cui scrivere i dati; ebbene, utilizzeremo l'handle dello StdOut con questa funzione per scrivere i nostri dati sullo StdOut stesso.

Alle righe 6-8 sono dichiarate tre [costanti](#) API per i tre stream (canali) standard del C: **STD_INPUT_HANDLE** per lo StdIn, **STD_OUTPUT_HANDLE** per lo StdOut e **STD_ERROR_HANDLE** per lo StdErr. Nel nostro esempio utilizzeremo soltanto lo StdOut; le altre due costanti sono specificate soltanto come riferimento.

Alla riga 10 abbiamo una variabile di nome **hConsole**; essa verrà utilizzata per memorizzare l'handle dello StdOut una volta ritrovato.

```
12. Private Sub Form_Load()  
13.     hConsole = GetStdHandle(STD_OUTPUT_HANDLE)  
14. End Sub  
15.
```

Al caricamento del form viene ritrovato l'handle dello StdOut mediante utilizzo della funzione GetStdHandle con la costante **STD_OUTPUT_HANDLE**. L'handle ritrovato verrà salvato nella variabile hConsole per essere utilizzato nelle operazioni di scrittura e chiusura della Console o dello StdOut.

```
16. Private Sub Form_Unload(Cancel As Integer)  
17.     Call CloseHandle(hConsole)  
18.     Call FreeConsole  
19. End Sub  
20.
```

Infatti alla chiusura del form viene richiamata la funzione CloseHandle con l'handle dello StdOut. Verrà anche chiusa l'eventuale Console aperta, mediante utilizzo della funzione FreeConsole.

```
21. Private Sub Testo_KeyPress(KeyAscii As Integer)  
22.     Dim Caratteri As Long  
23.     If KeyAscii = vbKeyReturn Then  
24.         Call WriteFile(hConsole, vbNewLine, 2, Caratteri, ByVal 0&)  
25.     Else  
26.         Call WriteFile(hConsole, Chr$(KeyAscii), 1, Caratteri, ByVal 0&)  
27.     End If  
28. End Sub
```

Il cuore di questo progetto si trova nella scrittura dei dati allo StdOut. Essa sarà assegnata alla pressione dei tasti nella *TextBox* **Testo**.

La variabile **Caratteri** dichiarata alla riga 22 servirà soltanto come valore di ritorno del numero di caratteri scritti per riferimento alla chiamata della funzione WriteFile.

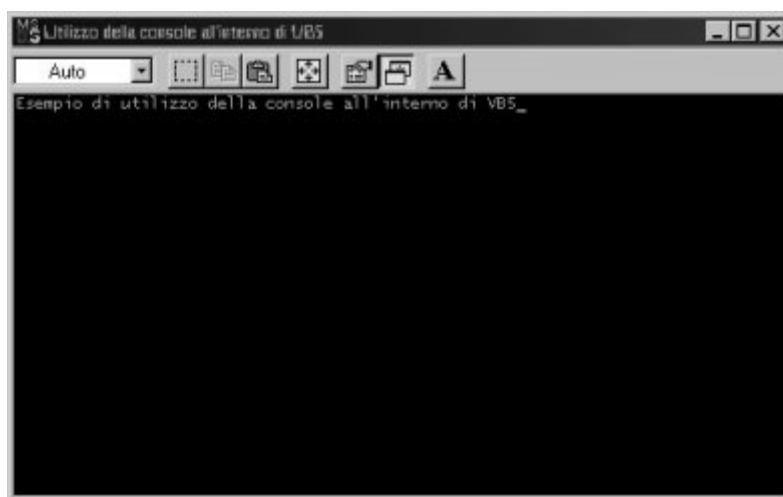
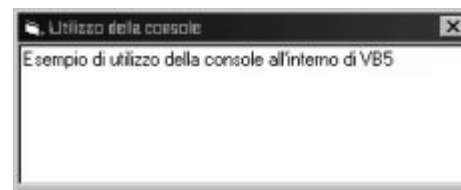
Alla riga 23 viene controllato se il tasto premuto è ENTER (costante *vbKeyReturn*): in tal caso dovrebbero essere mandati allo StdOut due caratteri, non uno, poiché nei sistemi Windows 9x l'ENTER è costituito dai due caratteri Carriage Return (costante *vbCr*) e Line Feed (costante *vbLf*).

Pertanto, se il tasto è ENTER, sarà richiamata la funzione WriteFile, con l'handle dello StdOut, la stringa *vbNewLine* (l'ENTER esteso) ed i caratteri da scrivere saranno 2. È necessario passare anche una variabile indicante il numero di caratteri scritti e sarà passata la variabile **Caratteri** (riga 24).

Se invece il tasto premuto non è ENTER, dovrà essere mandato allo StdOut il carattere del testo premuto ottenuto mediante l'istruzione *Chr\$* e dovrà essere scritto soltanto un

carattere (riga 26).

Il nostro progetto si conclude qui e possiamo provare il programma. Se il programma viene avviato all'interno dell'[IDE](#) sarà necessario richiedere l'apertura della Console, altrimenti tutti i dati scritti andranno persi poiché l'IDE non legge i dati inviati allo StdOut.



Si consiglia di compilare il progetto in programma EXE e richiamare il programma mediante riga di comando specificando anche il redirectionamento su un file o sulla stampante come segue: `CONSOLE > NOMEFILE`

L'utilizzo migliore si può avere utilizzando un'applicazione che legge i dati dallo StdOut come un browser, ma è necessario possedere anche un Web Server.

In tal caso copiare il file compilato nella cartella CGI-BIN del Web Server e richiamare il programma come segue: `http://indirizzoserver/cgi-bin/console.exe`

Tutto ciò che sarà digitato nella casella di testo sarà poi reinviato all'applicazione host che utilizza lo StdOut.

Il progetto così com'è presenta numerosi errori e limitazioni: il più pesante riguarda la Console aperta; se l'utente chiude la Console il programma viene chiuso con un errore. Se il progetto è eseguito all'interno dell'IDE di Visual Basic l'errore chiuderà anche l'IDE stesso.

Gli altri limiti riguardano il redirectionamento dell'output. Infatti se il programma compilato viene eseguito in un Prompt di MS-DOS l'output non sarà inviato alla console stessa.

Non sarà neanche possibile redirigerlo verso il dispositivo Console **CON** mediante `CONSOLE > CON`

Inoltre lo stream verrà interrotto subito dopo il richiamo del programma, poiché il processo viene eseguito in maniera [asincrona](#) rispetto al Prompt di MS-DOS. Ciò significa che l'utilizzo di eventuali Pipe genera errori poiché il buffer risulterà vuoto, ma in realtà sarà riempito solo in seguito.

Concludiamo il discorso che queste ultime due limitazioni non riguardano l'utilizzo del programma in applicazioni host quale un browser.

[Fibia FBI](#) e [Hal1961](#)

20 Maggio 2001



[Torna all'indice degli HowTo](#)
