

[Home Page](#) [Novità](#) [Aiuto](#) 

Sincrono o asincrono?

http://www.vbsimple.fbi/weird/wrd_08.htm

Difficoltà:  2 / 5

È risaputo che l'esecuzione di un'attività all'interno di un processo può essere **sincrona**, cioè l'esecuzione del codice successivo alla chiamata verrà eseguito soltanto quando la chiamata terminerà la sua attività e restituirà il controllo al chiamante, oppure **asincrona**, cioè le due attività proseguono indisturbate ed il chiamante non deve attendere la fine dell'attività del codice richiamato.

Quasi tutte le funzioni di Visual Basic sono sincrone e pertanto è possibile avere un certo grado di sicurezza; ad esempio la richiesta di creare una cartella con *MkDir* ci offre una certa sicurezza che al momento in cui verrà eseguita la riga successiva, salvo errori, la cartella sarà già stata creata. Infatti la chiamata alla funzione genera una chiamata ad un'altra funzione del sistema operativo, questo creerà la cartella alla posizione specificata e la funzione ritornerà al programma chiamante. Il codice successivo potrà tranquillamente utilizzare la suddetta cartella e non attendere che il sistema operativo la crei. Se per assurdo la creazione della cartella richiedesse un'ora di tempo il codice successivo alla funzione *MkDir* non sarà eseguito fino a quando la cartella non sarà creata ovvero dovrà attendere almeno un'ora.

Un esempio di funzione asincrona invece è *Shell* che lancia l'[esecuzione di un processo esterno](#) e ritorna immediatamente, il processo esterno si avvia e funziona in contemporanea al programma VB che l'ha richiamato. Il codice successivo alla funzione *Shell* sarà richiamato subito dopo la chiamata alla suddetta funzione, senza che sia necessario attendere la chiusura del processo esterno pertanto non si dovranno assumere posizioni certe su quel processo in esecuzione, dato che al momento del controllo potrebbe non essersi ancora avviato oppure potrebbe essere stato chiuso.

Durante l'attività di codice sincrone non si verificano tutti gli eventi che normalmente si generano e solitamente l'interfaccia grafica si blocca e lo schermo non viene neanche ridisegnato. È possibile provare questo con un semplice ciclo che duri un paio di secondi. Per tutto quel tempo non sarà possibile spostare il form, nè cliccare sui controlli in esso contenuti. Al termine del ciclo scatteranno tutti gli eventi in sospenso. Una deroga a questo comportamento è data dalla funzione *DoEvents* che forza l'esecuzione degli eventi in sospenso, arrestando l'attività al punto in cui si trova la chiamata alla funzione e lasciando spazio agli eventi in sospenso. Al termine dell'attività di tutti gli eventi in sospenso l'esecuzione del codice interrotto riprenderà normalmente. Per questa ragione la presenza di una funzione *DoEvents* all'interno di un ciclo lungo ne rallenta molto l'esecuzione, anche solo per ridisegnare lo schermo (attività questa molto lunga).

Questo è il comportamento normale di tutte le funzioni VB; essere sincrone o asincrone. In questo articolo invece si metterà [nuovamente](#) in analisi la funzione *SHFileOperation* che richiama le operazioni standard sui files, quali la copia, lo spostamento e la cancellazione, utilizzando l'interfaccia grafica che verrebbe mostrata se l'operazione venisse svolta da un

operatore, ovvero sarà visualizzata la finestra con i foglietti svolazzanti, la barra di avanzamento ed il pulsante di annullamento come mostrato nella **Figura 1**.

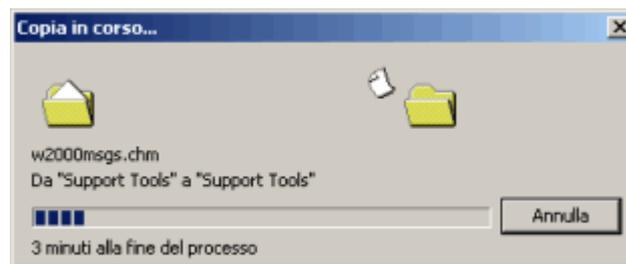
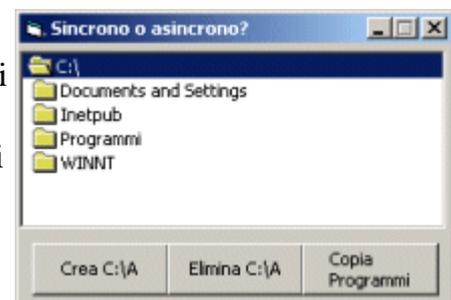


Figura 1

La funzione **SHFileOperation** si comporta in una maniera davvero strana: la sua attività non è definibile né sincrona né asincrona in quanto racchiude le caratteristiche di entrambe le modalità. Vediamo subito il progetto in analisi.

Utilizzeremo un unico form con quattro semplici controlli: una **DirListBox**  e tre **CommandButton**  dai rispettivi nomi **dirCartelle**, **cmdCreaA**, **cmdEliminaA** e **cmdCopiaProg**. Prima di vedere l'attività dei vari controlli inseriremo le solite dichiarazioni di costanti, funzioni e tipi utilizzati dall'API.



```

1. Option Explicit
2.
3. Private Const FO_COPY As Long = &H2
4. Private Const FO_DELETE As Long = &H3
5. Private Const FOF_ALLOWUNDO As Long = &H40
6.
7. Private Declare Function SHFileOperation Lib "shell32.dll" Alias
   "SHFileOperationA" (lpFileOp As SHFILEOPSTRUCT) As Long
8.
9. Private Type SHFILEOPSTRUCT
10.     hwnd As Long
11.     wFunc As Long
12.     pFrom As String
13.     pTo As String
14.     fFlags As Long
15.     fAnyOperationsAborted As Long
16.     hNameMappings As Long
17.     lpszProgressTitle As String
18. End Type
19.
20. Private Sub Form_Load()
21.     dirCartelle.Path = "C:\\"
22. End Sub
23.

```

 Porre particolare attenzione alla struttura SHFILEOPSTRUCT poiché molti articoli sul Web sono basati su una specifica errata e solo in seguito corretta da Microsoft. Vedi l'articolo [Si, no, forse...](#) in questa stessa sezione.

Le tre costanti dichiarate in cima serviranno a specificare alla funzione SHFileOperation cosa eseguire: una copia o una cancellazione e se spostare nel cestino durante la cancellazione.

La funzione dei tre pulsanti sul form è davvero molto semplice:

```

24. Private Sub cmdCreaA_Click()
25.     On Error Resume Next
26.     MkDir "C:\a"
27.     dirCartelle.Refresh
28. End Sub
29.

```

Il primo semplicemente creerà la cartella C:\A e quindi aggiornerà il contenuto dell'elenco **dirCartelle**. L'istruzione `On Error Resume Next` è stata aggiunta solo per semplificare il problema nel caso in cui la cartella esistesse già.

```

30. Private Sub cmdEliminaA_Click()
31.     Dim FileOperation As SHFILEOPSTRUCT
32.     Dim lReturn As Long
33.     With FileOperation
34.         .wFunc = FO_DELETE
35.         .pFrom = "C:\a"
36.         .fFlags = FOF_ALLOWUNDO
37.     End With
38.     lReturn = SHFileOperation(FileOperation)
39.     MsgBox "Completato"
40.     dirCartelle.Refresh
41. End Sub
42.

```

Il secondo pulsante eliminerà la cartella C:\A spostandola nel cestino (**FOF_ALLOWUNDO**) ed al termine dell'operazione mostrerà un avviso di completamento ed aggiornerà l'elenco **dirCartelle**.

```

43. Private Sub cmdCopiaProg_Click()
44.     Dim FileOperation As SHFILEOPSTRUCT
45.     Dim lReturn As Long
46.     With FileOperation
47.         .wFunc = FO_COPY
48.         .pFrom = "C:\Programmi"
49.         .pTo = "C:\a"
50.         .fFlags = FOF_ALLOWUNDO
51.     End With
52.     lReturn = SHFileOperation(FileOperation)
53.     MsgBox "Completato"
54.     dirCartelle.Refresh
55. End Sub

```

Il terzo pulsante, molto simile al secondo, copierà la cartella C:\Programmi in C:\A e come in precedenza, al termine dell'operazione mostrerà un avviso di completamento ed aggiornerà l'elenco **dirCartelle**.

Secondo quanto detto in precedenza la funzione *SHFileOperation* può definirsi (almeno in prima analisi) sincrona; infatti l'avviso di completamento non verrà eseguito fino a quando la funzione non ritornerà al programma che l'ha chiamata, ovvero fino a quando non avrà completato l'operazione. Possiamo semplicemente dimostrarlo lanciando il programma, creando la cartella con il primo dei tre pulsanti e cancellarla con il secondo.

Sarà mostrata la richiesta come nella **Figura 3** e solo dopo aver risposto SI alla domanda la cartella verrà cancellata e l'avviso di completamento sarà mostrato.

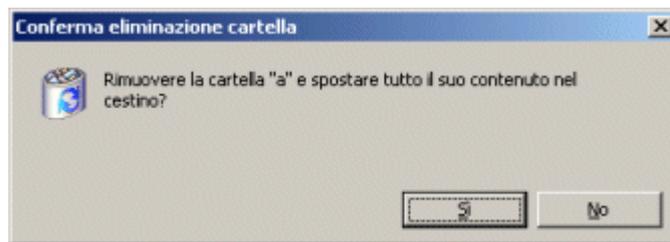


Figura 3



Figura 4

Il fatto che l'avviso della **Figura 4** non venga mostrato fino a quando l'attività della funzione *SHFileOperation* non viene completata dimostra che la funzione è sincrona, cioè arresta il codice che l'ha chiamata fino a quando non termina la sua attività.

È possibile provare la stessa operazione cliccando sul terzo pulsante "**Copia Programmi**"; anche in questo caso la funzione si comporta in maniera modale. Per tutta la durata dell'operazione la routine che ha chiamato la funzione è sospesa e la finestra di completamento non verrà mostrata. Solo al termine dell'operazione oppure all'interruzione da parte dell'utente cliccando sul pulsante Annulla il codice proseguirà da dove era stato interrotto. Questa è il normale funzionamento delle funzioni sincrone.

Tuttavia la funzione nasconde un problema di non poco conto: proviamo a lanciare l'operazione di copia mediante il terzo pulsante e durante la sua attività (è stata scelta la cartella programmi perché la durata dell'operazione è abbastanza lunga), cioè mentre è mostrata la finestra di avanzamento, possiamo cliccare sulla finestra del nostro progetto e trovarla completamente funzionante. Possiamo spostarla, chiuderla, cliccare sui pulsanti e persino sul terzo pulsante, cioè quello che ha richiamato la funzione di copia, ancora in corso.

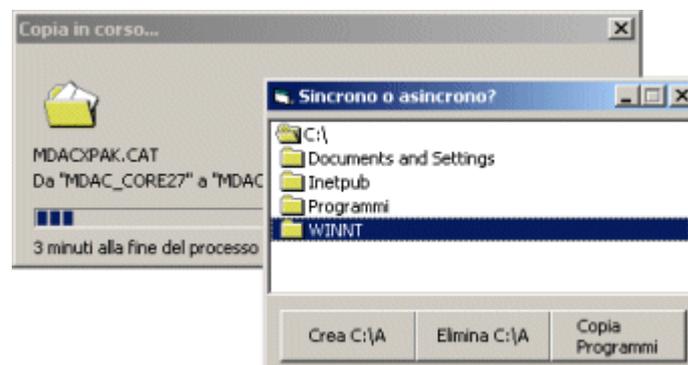


Figura 5

Questo è invece il tipico funzionamento delle funzioni asincrone, che lasciano il controllo al programma chiamante anche durante la loro attività. Questo genere di comportamento può condurre a parecchi errori poiché l'attività della funzione di copia non è stata ancora completata ma è comunque possibile tentare la cancellazione della cartella.

Ma come abbiamo già visto la funzione non è asincrona; infatti al termine oppure all'interruzione mediante il pulsante Annulla, sarà eseguito il codice successivo alla chiamata della funzione e quindi verrà mostrato l'avviso di completamento.

Dovendo dare una spiegazione di questo particolare comportamento della funzione *SHFileOperation* si può dire che all'interno del ciclo di copiatura è presente una chiamata ad una funzione analoga a *DoEvents*, che consenta l'esecuzione degli eventi pendenti, dal

semplice aggiornamento grafico della finestra al completo funzionamento degli altri controlli sullo schermo.

[Fibia FBI](#)

15 Febbraio 2004



[Torna all'indice delle Stranezze](#)
