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

Inserire un pulsante sulla barra del titolo 

[http://www.vbsimple.net/howto/ht\\_022.htm](http://www.vbsimple.net/howto/ht_022.htm)

Difficoltà:  4 / 5

Avete mai pensato di inserire un pulsante sulla barra del titolo di un form?  
Alcuni programmi già lo fanno, ma sono soltanto tre o quattro. Come realizzare tutto ciò?

Il problema in questione è risolvibile solamente con un'operazione di [subclassing](#) di un semplice *CommandButton*. Infatti è impossibile spostare un pulsante sulla barra del titolo tramite le normali istruzioni di Visual Basic e, anche se riuscissimo a farlo, non esiste un evento del form che rilevi il suo spostamento.

Pertanto, tramite una funzione di [hooking](#), rintracceremo i [messaggi](#) diretti alla finestra del pulsante e in tal modo potremo eseguire operazioni particolari legate all'invio di messaggi non legati ad eventi dell'oggetto subclassato.

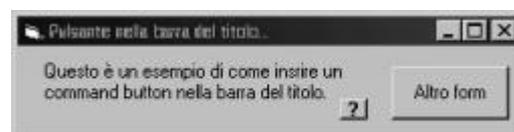
### Attenzione!

Si raccomanda di non interrompere l'esecuzione del programma tramite le funzioni di Visual Basic o attraverso l'istruzione END prima di aver disinstallato la *Window Procedure* ed aver ripristinato lo stato normale delle finestre. Interrompere l'esecuzione del programma mantenendo la *Window Procedure* causerà un errore del programma ed i rischi connessi potrebbero essere imprevedibili.

In questo progetto avremo bisogno di due forms  e di un [modulo](#)  standard per le dichiarazioni [API](#).

Cominciamo disegnando l'interfaccia grafica: il primo form si chiamerà **frmMain** e sarà quello che conterrà il pulsante che prenderà posto sulla barra del titolo.

Il form conterrà una *Label*  descrittiva senza alcuna funzionalità e due *CommandButton*  di cui il primo si chiamerà *cmdTest* ed avrà queste proprietà:



- *Font*: Ms Sans Serif 8 in Grassetto
- *Caption*: "?"
- *Height*: 195
- *Width*: 255

Il secondo pulsante si chiamerà **Command1** e conterrà il testo "*Altro form*". Servirà a caricare il secondo form e dimostrare il comportamento del primo pulsante in presenza di altri forms.

Disegniamo rapidamente anche l'altro form, di nome **Form1**. Esso conterrà soltanto un *CommandButton* di nome **Command1**, con la *Caption* impostata a "*Chiudi*". Questo

secondo form non farà nulla; il suo utilizzo serve soltanto per disattivare il primo form.

---

Passiamo al codice. Dentro il modulo  standard scriviamo:

```

1. Option Explicit
2.
3. Public Declare Function GetWindowRect Lib "user32" (ByVal hwnd As Long, lpRect As Rect) As Long
4. Public Declare Function GetParent Lib "user32" (ByVal hwnd As Long) As Long
5. Public Declare Function SetParent Lib "user32" (ByVal hWndChild As Long, ByVal hWndNewParent As Long) As Long
6. Public Declare Function SetWindowPos Lib "user32" (ByVal hwnd As Long, ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As Long
7. Public Declare Function SetWindowsHookEx Lib "user32" Alias "SetWindowsHookExA" (ByVal idHook&, ByVal lpfn&, ByVal hmod&, ByVal dwThreadId&) As Long
8. Public Declare Function UnhookWindowsHookEx Lib "user32" (ByVal hHook&) As Long
9. Public Declare Function SetWindowLong Lib "user32" Alias "SetWindowLongA" (ByVal hwnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

```

Fino a qui abbiamo le varie dichiarazioni API:

la funzione *GetWindowRect* serve per ottenere le coordinate assolute del form sullo schermo;

la funzione *GetParent* restituisce l'[handle](#) della finestra genitore di una finestra specifica;

la funzione *SetParent* invece permette di cambiare la finestra genitore di una certa finestra;

la funzione *SetWindowPos* permette di cambiare la posizione di una finestra;

le due funzioni *SetWindowsHookEx* e *UnhookWindowsHookEx* permettono di agganciare e sganciare una funzione e creare così una *Window Procedure*;

infine, l'ultima funzione, *SetWindowLong*, serve per eseguire molteplici funzioni: nel nostro caso la utilizzeremo per cambiare lo stile della finestra del pulsante da spostare.

Infatti questa finestra verrà cambiata di famiglia; dallo stato di child del form diventerà una finestra del tutto autonoma, allo stesso livello del form. La funzione *SetWindowLong* servirà per nascondere la finestra del pulsante dalla barra delle applicazioni.

```

10. Public Type Rect
11.     Left As Long
12.     Top As Long
13.     Right As Long
14.     Bottom As Long
15. End Type
16.

```

Il tipo *Rect* viene richiesto dalla funzione *GetWindowRect* per ottenere le coordinate assolute della finestra.

```

17. Public Type CWPSTRUCT
18.     lParam As Long
19.     wParam As Long
20.     Message As Long
21.     hwnd As Long
22. End Type
23.

```

Il tipo *CWPSTRUCT* - *CallWindowProcedureStruct* - viene utilizzato dalla nostra *Window Procedure* per contenere i parametri passati dalla *Window Procedure* originale.

```

23. Public Const WM_MOVE = &H3
24. Public Const WM_NCPAINT = &H85
25. Public Const WM_COMMAND = &H111
26. Public Const SWP_FRAMECHANGED = &H20
27. Public Const WH_CALLWNDPROC = 4
28. Public Const WS_EX_TOOLWINDOW = &H80
29. Public Const GWL_EXSTYLE = -20
30.
31. Public oldproc As Long
32. Public cmdButtonHwnd As Long
33.

```

Seguono alcune [costanti](#) per definire i comportamenti da effettuare tramite le funzioni [API](#) viste poco prima. Le costanti alle righe 23, 24 e 25 sono i tre [messaggi](#) che intercetteremo tramite la *Window Procedure* e in base ad essi svogheremo determinate operazioni.

La variabile **oldproc** (riga 31) contiene il puntatore alla locazione della *Window Procedure* originale. La variabile verrà utilizzata per ripristinare lo stato normale delle al termine del programma.

La variabile **cmdButtonHwnd** contiene l'[handle](#) del pulsante e verrà utilizzata da alcune funzioni API per decidere la posizione che la finestra del pulsante deve assumere.

```

34. Public Function CallWndProc(ByVal nCode As Long, ByVal wParam As Long, Inf As
    CWPSTRUCT) As Long
35.     Dim FormRect As Rect
36.     Select Case Inf.Message
37.         Case WM_COMMAND
38.             If Inf.lParam = cmdButtonHwnd Then
39.                 frmMain.cmdTest.Visible = False
40.                 MsgBox "Per ulteriori info su questo esempio:
sirri@morenosoft.com", vbInformation, frmMain.Caption
41.                 frmMain.cmdTest.Visible = True
42.             End If
43.         Case WM_NCPAINT, WM_MOVE
44.             Call GetWindowRect(frmMain.hwnd, FormRect)
45.             Call SetWindowPos(cmdButtonHwnd, 0, FormRect.Right - 75, FormRect.Top +
    6, 17, 14, SWP_FRAMECHANGED)
46.         End Select
47. End Function

```

Alla riga 34 inizia la nostra *Window Procedure*, di nome **CallWndProc**. Essa dovrà processare tre messaggi: WM\_COMMAND, WM\_NCPAINT e WM\_MOVE.

Così, alla riga 36, abbiamo la scelta del messaggio passato. Se esso è WM\_COMMAND, ovvero la pressione del pulsante (riga 38), esso verrà prima nascosto (riga 39), sarà mostrata una finestra con un messaggio (riga 40) e poi sarà nuovamente reso visibile (riga 41).

Se il messaggio passato alla *Window Procedure* è WM\_NCPAINT (ridisegna la finestra) oppure WM\_MOVE (finestra spostata), verranno estratte le coordinate assolute del form **frmMain** tramite la funzione *GetWindowRect* (riga 44) e poi verrà spostato il pulsante il cui handle è **cmdButtonHwnd** (ovvero il pulsante **cmdTest**) tramite la funzione *SetWindowPos*. Viene effettuato qualche calcolo per determinare la posizione che il pulsante dovrà assumere sulla barra del titolo (riga 45).

Tutti gli altri messaggi inviati alla finestra del form saranno ignorati.

Torniamo al form principale (**frmMain**) e scriviamo queste righe di codice:

```
1. Option Explicit
2.
3. Private Sub Form_Load()
4.     cmdButtonHwnd = cmdTest.hwnd
5.     oldproc = SetWindowsHookEx(WH_CALLWNDPROC, AddressOf CallWndProc, 0,
App.ThreadID)
6.     Call SetWindowLong(cmdButtonHwnd, GWL_EXSTYLE, WS_EX_TOOLWINDOW)
7.     Call SetParent(cmdButtonHwnd, GetParent(frmMain.hwnd))
8. End Sub
9.
```

L'hooking della finestra viene effettuato in occasione del caricamento del Form, all'interno dell'[evento](#)  Load.

Alla riga 4 viene salvato l'handle del pulsante **cmdTest**, per riutilizzarlo in seguito all'interno del modulo.

Alla riga 5 viene installata la nuova Window Procedure. La funzione *SetWindowsHookEx* richiede fra l'altro, un puntatore alla funzione Window Procedure. Esso viene fornito tramite l'operatore *AddressOf*. La costante *WH\_CALLWNDPROC* indica che la nostra Window Procedure riceverà i messaggi **prima** che il sistema li invii alla Window Procedure originale.

L'indirizzo della Window Procedure originale viene comunque salvato nella variabile **oldproc** come valore di ritorno della funzione. Esso servirà per ripristinare lo stato normale delle finestre al termine dell'esecuzione del programma.

La riga 6 cambia lo stile della finestra del pulsante, trasformandola in una finestra invisibile nella barra delle applicazioni, perché alla prossima riga la finestra del pulsante diventerà un'applicazione vera e propria, non sottomessa ai limiti imposti dal form.

Alla riga 7 viene cambiata la finestra genitore del pulsante *cmdTest*, e viene impostata come finestra genitore la finestra contenente il form, ovvero il Program Manager stesso, il programma che gestisce le finestre di tutti gli altri programmi.

Ecco perché abbiamo dovuto cambiare lo stile della finestra alla riga 6. Se non l'avessimo fatto avremmo visto l'icona del pulsante ? sulla barra delle applicazioni.

```
10. Private Sub Form_Unload(Cancel As Integer)
11.     Call UnhookWindowsHookEx(oldproc)
12.     Call SetParent(cmdButtonHwnd, frmMain.hwnd)
13. End Sub
14.
```

Alla chiusura del form, nell'[evento](#)  Unload, abbiamo lo sganciamento (*unhooking*) della Window Procedure, tramite la chiamata della funzione *UnhookWindowsHookEx* con il parametro della Window Procedure originale.

Viene anche ripristinata la finestra genitore del pulsante, reimpostandogli come genitore il form *frmMain*.

```
15. Private Sub cmdTest_Click()
```

```
16.     MsgBox "Questo evento non verrà eseguito mai!"
17. End Sub
18.
```

Abbiamo voluto aggiungere questa piccola routine per dimostrare che la finestra del pulsantino **cmdTest** diviene un programma del tutto separato. Infatti al click sopra il pulsante non viene generato più l'evento Click, gestito dalla nostra applicazione.

L'esecuzione di questo evento non avverrà mai.

```
19. Private Sub Command1_Click()
20.     cmdTest.Visible = False
21.     Load Form1
22.     Form1.Show
23. End Sub
24.
```

Alla riga 19 abbiamo gestito l'evento Click del pulsante Command1. Esso inizialmente nasconderà il pulsante sulla barra del titolo (riga 20), caricherà e mostrerà il secondo form, di nome Form1. Questa funzione serve soltanto a dimostrare il funzionamento del pulsante sulla barra del titolo in presenza di più forms nella stessa applicazione.

```
25. Private Sub Form_Activate()
26.     cmdTest.Visible = True
27. End Sub
28.
```

Nel momento in cui la finestra principale viene attivata, torna visibile il pulsante cmdTest.

```
29. Private Sub Form_Deactivate()
30.     cmdTest.Visible = False
31. End Sub
32.
```

E nel momento in cui la finestra perde lo stato attivo, il pulsante viene nascosto.

Questo viene effettuato perché la gestione grezza dei messaggi fa sì che il nostro pulsante sia in primo piano pure quando la finestra che sembra contenerlo (in effetti esso è diventato finestra autonoma) passa in secondo piano.

---

Prima di provare il programma scriviamo queste semplici righe di codice per il secondo form di nome Form1.

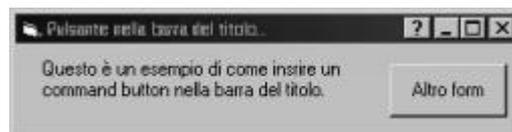
```
1. Option Explicit
2.
3. Private Sub Command1_Click()
4.     Unload Me
5. End Sub
```

La sua unica funzione è quella di chiudere il form nel momento in cui l'utente clicca sul pulsante **Command1**.

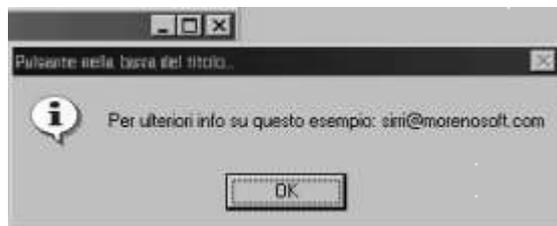
---

Possiamo adesso provare il programma.

All'esecuzione il pulsantino ? si sposterà alle stesse coordinate dei pulsanti di chiusura e ridimensionamento del form.



In realtà sappiamo che è un trucchetto ottico e la situazione non sta proprio così.



**Figura 3**

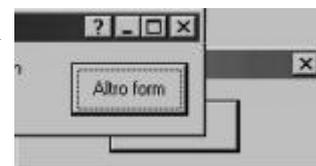
Il click sopra il pulsantino fa apparire una finestra con un messaggio. Come effetto collaterale, però, vediamo che il pulsante ? è sparito dalla barra del titolo del form su cui sembrava posizionato.

La stessa cosa accade quando l'utente clicca il pulsante "**Altro form**" che mostra il secondo form.

Quando il secondo form è in primo piano (e il primo form è disattivo) il pulsante sulla barra del titolo del primo sparisce.



Basterà riattivare la finestra del primo form, anche senza chiudere il secondo form, per vedere ritornare presente il pulsante sulla barra del titolo.



Sappiamo che è uno strano giochetto, ma è stato reso necessario per evitare spiacevoli comportamenti del pulsante.

Questo genere di operazioni potrà sembrare molto carino, anche se un po' complesso, ma nasconde decine di insidie.

In ogni caso non fermare mai l'esecuzione del programma tramite la voce Interrompi dell'[IDE](#) di Visual Basic, oppure attraverso l'istruzione END. Ciò lascerebbe l'hook in memoria e la Window Procedure non verrebbe rimessa al suo posto.

Questo genererà subito un errore che chiude inesorabilmente l'editor di Visual Basic e può anche trascinarsi dietro un blocco del sistema.

In definitiva le operazioni di subclassing devono essere adoperate il meno possibile e prima di provare i progetti che le utilizzano salvare tutti i lavori. Particolare cura deve essere data pure alle operazioni di chiusura del programma, in modo da sganciare tutti gli hook eseguiti all'avvio del programma.

[Moreno Sirri](#)  
22 Gennaio 2001

 [Torna all'indice degli HowTo](#)