Home Page
Informazioni
Aiuto

Invio dati ad un'applicazione esterna

http://www.vbsimple.net/activity/act_23.htm

Richiesta di: LoneFlier - 2 Settembre 2002
Difficoltà: 4 / 5

Ho scritto un programma in VB [...] non riesco però a fare in modo che, dopo essere ridotto ad icona, mi invii i dati all'applicazione attiva, che non dovrebbe essere definita dal programma.

La questione non è molto semplice come appare in realtà. Infatti se l'utente va a richiamare un programma esterno, l'applicazione scritta con VB, ridotta ad icona, non è più la finestra attiva.
È pertanto necessario che scatti qualche evento che dica al programma ***"ok, ho scelto questa applicazione, mandami i dati"***. Non si potrà naturalmente cliccare un pulsante sul programma VB, altrimenti quest'ultimo diverrà l'applicazione attiva ed i dati saranno inviati nuovamente al suddetto programma.

Assumiamo che il punto precedente si superi impostando un certo ritardo prima dell'invio di dati (ad esempio 5 secondi), poiché una finestra di Windows si compone di tantissime finestre client (le caselle di testo, i pulsanti) è naturale che l'utente, nella scelta dell'applicazione attiva debba scegliere **esattamente** il controllo in cui andranno inviati questi dati. Un'applicazione spartana come il *Notepad* ha due sole finestre, la parent (Application Window) che contiene tutto (compresi i menu) e la grande casella di testo. Un'applicazione più complessa come *MS Word* contiene centinaia di finestre (l'area grande in cui scrivere il testo, i pulsanti per grassetto, corsivo, le caselle combo in cui si seleziona la dimensione o lo stile del carattere, etcc..) e ciò obbliga l'utente ad una scelta precisa.
[In risposta a LoneFlier il 5 Settembre 2002]

In questo articolo faremo uso di *Notepad*, una semplicissima applicazione esterna che si compone di due sole finestre. È fondamentale che l'utente o il programma sappiano esattamente a quale finestra dovranno essere inviati i dati. Esiste un'altra soluzione un po' più precisa per la decisione della finestra che riceverà i dati e sarà trattata nella seconda parte di quest'articolo.

Nella soluzione sviluppata in quest'articolo sarà quindi compito dell'utente cliccare sopra la finestra che riceverà i dati, entro un certo limite di tempo, deciso in 2 secondi.

Il punto chiave del problema sta nel recupero della finestra attiva in cui si trova il cosiddetto focus di tastiera ovvero quella finestra che riceverebbe dati se l'utente premesse uno o più tasti sulla tastiera; si tratta in fondo del controllo che ha lo stato attivo. Purtroppo non esiste alcuna funzione nativa in grado di recuperare tale informazione; utilizzeremo pertanto una funzione di nome ***GetFocusEx*** ma attenzione che nonostante il nome **non si tratta di una funzione API**, ma di una nuova funzione definita all'interno del nostro form;

l'autore della funzione è Fabio Zanetta (Zanna) il quale ne ha concesso l'utilizzo in questo tutorial.

Il progetto si compone di un unico form con due soli controlli: una *TextBox* di nome **txtTesto** ed un *CommandButton* di nome **cmdInvia**. Il pulsante naturalmente servirà per inviare i dati all'applicazione esterna Notepad.
Il codice si presenta ricco di funzioni API:

```
1. Option Explicit
2.
3. Private Declare Function GetForegroundWindow Lib "user32" () As Long
4. Private Declare Function SetForegroundWindow Lib "user32" (ByVal hwnd As Long) As
   Long
5. Private Declare Function GetFocus Lib "user32" () As Long
6. Private Declare Function AttachThreadInput Lib "user32" (ByVal idAttach As Long,
   ByVal idAttachTo As Long, ByVal fAttach As Long) As Long
7. Private Declare Function GetWindowThreadProcessId Lib "user32" (ByVal hwnd As Long,
   lpdwProcessId As Long) As Long
8.
```

Questa prima serie di funzioni API riguardano il recupero della finestra attiva: le funzioni **GetForegroundWindow** e **SetForegroundWindow** servono rispettivamente per recuperare l'handle della finestra **Parent** in primo piano e per cambiare la finestra in primo piano.

La funzione **GetFocus** restituisce l'handle della finestra con il focus di tastiera attivo, ma soltanto all'interno del thread da cui si richiama la funzione. Sarebbe perfetta se recuperasse l'handle della finestra attiva da qualsiasi thread, ma purtroppo così non fa; riporta soltanto la finestra con il focus di tastiera all'interno del thread da cui si richiama; è però possibile *ingannare* la funzione collegando i due thread in uno solo; GetFocus quindi riconoscerebbe la finestra attiva (la casella di testo all'interno di Notepad) come parte del thread del programma VB.

Il collegamento dei due thread è possibile mediante la funzione **AttachThreadInput** che richiede l'identificativo dei due thread da collegare. È possibile recuperare i due thread mediante la funzione **GetWindowThreadProcessId** che restituisce il thread dentro al quale è stata creata la finestra specificata. Queste funzioni interconnesse tra loro saranno richiamate all'interno della funzione ***GetFocusEx*** che vedremo più avanti.

```
 9. Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
10. Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" (ByVal hwnd
    As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
11.
12. Private Const WM_CHAR = &H102
13. Private Const WM_SETTEXT = &HC
14. Private Const EM_SETSEL = &HB1
15.
```

La Sub **Sleep**, descritta in altro articolo, serve esclusivamente per forzare un'attesa del codice di un certo tempo. Nel nostro caso servirà per permettere all'utente la scelta della finesta a cui inviare i dati.

Infine la funzione **SendMessage**, anche questa vista in tanti altri articoli, consente di inviare un messaggio ad una funzione specifica. Il messaggio sarà quindi elaborato dalla

relativa Window Procedure che reagirà di conseguenza.

I messaggi che invieremo alla finestra scelta sono quelli definiti alle righe 12-14 e servono rispettivamente ad inviare un carattere, ad inviare un testo ed a modificare la posizione del cursorse della finestra indicata.

```
16. Private Sub Form_Load()
17.     Call Shell("NOTEPAD.EXE", vbNormalNoFocus)
18. End Sub
19.
```

Il progetto si apre con il richiamo del processo esterno NOTEPAD.EXE, che avvierà un'istanza del già noto Blocco Note di Windows. Il parametro ***vbNormalNoFocus*** farà sì che il processo venga avviato in secondo piano e la nuova finestra non prenda il sopravvento sul nostro progetto VB.

Seguitiamo vedendo il codice della funzione ***GetFocusEx***, di cui abbiamo già accennato il funzionamento:

```
20. Private Function GetFocusEx(ByVal Child As Boolean) As Long
21.     Dim t_id As Long
22.     Dim c_t_id As Long
23.     If Child Then
24.         t_id = App.ThreadID
25.         c_t_id = GetWindowThreadProcessId(GetForegroundWindow, 0&)
26.         Call AttachThreadInput(c_t_id, t_id, True)
27.         GetFocusEx = GetFocus()
28.         Call AttachThreadInput(c_t_id, t_id, False)
29.     Else
30.         GetFocusEx = GetForegroundWindow()
31.     End If
32. End Function
33.
```

La funzione richiede il passaggio di un argomento: Child di tipo Booleano specifica se è richiesto il recupero della finestra Client o di quella Parent attiva. Nel nostro caso è chiaro che ci interessa sapere quale controllo ha lo stato attivo, ovvero quale finestra child ha il focus di tastiera.

Alle righe 21 e 22 sono dichiarate due variabili di tipo Long: **t_id** specificherà il thread della nostra applicazione, mentre **c_t_id** conterrà il thread dell'applicazione esterna da collegare alla prima.
I due valori sono infatti recuperati alle righe 24 e 25: *App.ThreadID* specifica il thread della nostra applicazione Visual Basic; il secondo thread è recuperato mediante **GetWindowThreadProcessId**, fornendogli l'handle della finesta in primo piano (recuperato a sua volta tramite **GetForegroundWindow**). Invece di *App.ThreadID* avremmo potuto richiamare la stessa funzione, fornendogli l'handle del form da cui parte l'elaborazione.

Ottenuti i due thread sarà quindi possibile collegarli tra loro mediante **AttachThreadInput**; il primo argomento della funzione specifica il thread da collegare, mentre il secondo argomento specifica il thread a cui collegare il primo; l'ultimo argomento infine determine se la funzione dovrà collegare o scollegare i due thread.

Le due operazioni saranno infatti eseguite alle righe 26 e 28. Subito dopo aver collegato i

due thread sarà possibile utilizzare la succitata funzione **GetFocus** che, non più dolente del difetto di partenza legato al thread da cui si esegue, sarà in grado di dirci quale finestra child ha il focus di tastiera. Recuperato tale valore i due thread saranno quindi scollegati.

Nel caso che la funzione ***GetFocusEx*** venga richiamata con il parametro *Child* False, sarà recuperata la finestra Parent attiva mediante **GetForegroundWindow** (riga 30).

Attenzione!
La funzione ***GetFocusEx*** nei sistemi Windows 9x (soprattutto in Windows 98), per ragioni ignote, a volte fallisce il suo scopo e restituisce valore 0.
A tal scopo, vedremo in seguito, verrà aggiunto un controllo sul valore restituito dalla funzione e l'operazione sarà ritentata in caso di valore 0.

Il resto del progetto riguarda la gestione dell'evento Click sopra il pulsante **cmdInvia**.

```
34. Private Sub cmdInvia_Click()
35.     Dim lnghWnd As Long
36.     Dim lnghWndParent As Long
37.     Dim intConta As Integer
38.     Dim bytBuffer() As Byte
39.     Dim intRetries As Integer
40.     Const intMaxRetries = 5
41.     Const lngTypeDelay = 0
42.
```

La routine utilizza parecchie funzioni per avere il massimo controllo possibile sulla finesta a cui inviare i dati. La variabile **lnghWnd** conterrà l'handle della finestra child con il focus di tastiera attivo, restituito dalla funzione ***GetFocusEx***; **lnghWndParent** conterrà invece l'handle della finesta Parent attiva, ovvero di Notepad. Quindi lnghWnd sarà una finestra contenuta all'interno della finesta indicata da lnghWndParent.

La variabile **intConta** sarà utilizzata per alcuni cicli che vedremo in seguito; **bytBuffer** è una matrice che conterrà invece i dati da inviare all'applicazione; infine **intRetries** conterà il numero di tentativi in caso di fallimento della funzione ***GetFocusEx*** e **lngTypeDelay** determina il ritardo tra l'invio di ogni carattere.

Le due costanti definite alle righe 40 e 41 indicano rispettivamente il numero massimo di tentativi per la funzione ***GetFocusEx*** da eseguire prima di dichiarare fallita l'operazione, e la velocità di digitazione del testo nella finestra esterna. Sarà tutto più chiaro fra qualche riga.

```
43.     MsgBox "Premi OK e clicca su Notepad! (2 Secondi)", vbExclamation
44.     Call Sleep(2000)
45.     lnghWndParent = GetForegroundWindow()
46.     Do While (lnghWnd = 0) And (intRetries < intMaxRetries)
47.         lnghWnd = GetFocusEx(True)
48.         intRetries = intRetries + 1
49.         Call SetForegroundWindow(lnghWndParent)
50.     Loop
51.     If lnghWnd = 0 Then
52.         MsgBox "Impossibile attivare la finestra", vbCritical
53.         Exit Sub
54.     End If
55.
```

Le righe 43 e 44 informano l'utente che è giunto il momento di scegliere la finesta a cui

inviare i dati e nel nostro caso sarà la finestra di Notepad. Concederemo all'utente fino a 2 secondi (ovvero 2000 millisecondi) per scegliere la finestra ed in questo tempo il nostro programma rimarrà in attesa (riga 44).

Superato questo momento si procederà innanzitutto a recuperare l'handle della finestra Parent attiva (riga 45) e quindi a recuperare l'hande della finestra Child in essa contenuta. Come abbiamo già detto, poiché la funzione in alcune situazioni restituisce valore 0, alle righe 46-50 sarà eseguito un ciclo che si ripeterà fino a quando non verrà recuperato l'handle della finestra, fino ad un massimo di 5 volte (come specificato dalla costante **intMaxRetries** vista alla riga 40.

All'interno del ciclo sarà innanzitutto recuperata la finestra Child mediante GetFocusEx e, giusto per sicurezza, riportata la finestra Parent in primo piano. Infatti a volte la funzione ***GetFocusEx***, nel tentativo di recuperare l'handle della finestra disattiva o addirittura porta in secondo piano la finestra su cui lavora.

Se al termine del ciclo di controllo, il valore di **lnghWnd** sarà ancora 0, l'elaborazione sarà interrotta con un avviso per l'utente (righe 51-54).

Con l'handle della finestra di destinazione contenuto nella variabile ***lnghWnd*** possiamo finalmente passare al succo del problema: inviare il testo contenuto nella casella di testo alla finestra appena individuata.

```
56.      bytBuffer = StrConv(txtTesto.Text & vbCrLf & vbNullChar, vbFromUnicode)
57.      Call SendMessage(lnghWnd, WM_SETTEXT, ByVal 0&, bytBuffer(0))
58.      Call SendMessage(lnghWnd, EM_SETSEL, UBound(bytBuffer, 1), 0)
59.      For intConta = 0 To UBound(bytBuffer, 1) - 2
60.          Call SendMessage(lnghWnd, WM_CHAR, bytBuffer(intConta), ByVal 0&)
61.          If lngTypeDelay > 0 Then Call Sleep(lngTypeDelay)
62.      Next intConta
63.      Call SetForegroundWindow(lnghWndParent)
64.  End Sub
```

La prima operazione è la conversione in matrice di bytes della stringa da inviare ed è effettuata alla riga 56. Durante la conversione è stato aggiunto un ritorno a capo (giusto per l'estetica) ed un carattere NULL che indica la fine della stringa. La ragione è spiegata nell'articolo di approfondimento sullo studio delle stringhe API.

Finalmente alla riga 57 invieremo i dati alla finestra di destinazione, sotto forma di array di bytes, tramite la funzione **SendMessage** ed il messaggio ***EM_SETTEXT***, che corrisponde a ciò che in VB otterremmo con `Text1.Text = Stringa` ovvero tutto il testo precedente verrà eliminato e sostituito da quello inviato. Ma non è finita qui: con l'occasione abbiamo voluto mostrare anche una seconda soluzione ovvero quella di inviare la stringa carattere per carattere, come se i caratteri fossero battuti da tastiera e senza eliminare il testo precedente.

Alla riga 58 riposizioniamo il cursore alla fine del testo, in modo da poter inviare altri dati senza sovrascrivere o spostare quelli precedenti. Anche per questa funzione ci viene incontro **SendMessage** con il messaggio ***EM_SETSEL***.

Sarà infine eseguito un ciclo per tutti i caratteri contenuti nella matrice di bytes (ad eccezione del NULL) all'interno del quale sarà inviato l'ultimo messaggio ***WM_CHAR*** che

corrisponde alla pressione del tasto da tastiera; il codice del tasto sarà specificato nel terzo argomento alla funzione. Per simulare ancora meglio l'azione è stata aggiunta (riga 61) la possibilità di effettuare un piccolo ritardo tra ogni battuta e ciò è determinato dal valore della costante **lngTypeDelay**.

Per simulare un effetto gradevole si consiglia di impostare il valore della costante lngTypeDelay ad un valore compreso tra 100 e 500.

Possiamo quindi passare alla dimostrazione del progetto fino a qui sviluppato: all'avvio del progetto sarà eseguito anche Notepad che rimarrà in secondo piano. Inserire del testo nella casella di testo del form e premere il pulsante ***Invia***.

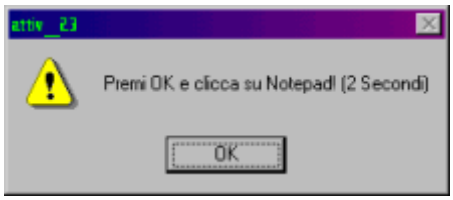

Figura 3

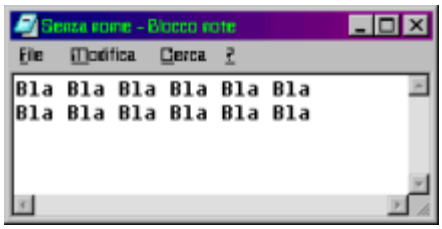

Figura 4

Sarà mostrato un avviso come quello presentato nella Figura 3 e l'utente avrà il limite di 2 secondi per attivare la finestra del Blocco Note a cui verranno inviati i dati (vedi Figura 4).

Il problema è alquanto complesso e la soluzione proposta non è delle ottimali perché troppo sofferente delle scelte dell'utente o dell'incapacità dell'utente o del sistema di attivare la finestra di destinazione nel tempo limite. In caso di una scelta errata i risultati possono essere imprevedibili.

Per ovviare a queste chiare difficoltà sarà proposta una seconda soluzione, molto più armoniosa e pulita di quella appena presentata. Si raccomanda pertanto la consultazione della seconda parte di questo articolo.

La presente soluzione è stata comunque sviluppata un po' per pretesto per entrare nel merito della complessità del problema e per comprendere i meccanismi che regolano le comunicazioni tra processi esterni.

Seconda Parte >>

Fibia FBI
5 Settembre 2002

Torna all'introduzione delle Richieste dei lettori

