



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

## Richiamo di funzioni mediante puntatore


[http://www.vbsimple.net/howto/ht\\_048.htm](http://www.vbsimple.net/howto/ht_048.htm)





Difficoltà:  3 / 5

Sviluppando un programma di medie dimensioni potrebbe rendersi necessario dover scegliere se eseguire una prima o una seconda funzione. In genere queste scelte sono effettuate con un semplice If..Then.

Se tuttavia stiamo scrivendo un codice ampio oppure sviluppando un componente in grado di utilizzare funzioni esterne poste in una [DLL](#) separata si rende impossibile la scelta tramite una selezione. In questo genere di situazioni si utilizzano i [puntatori](#) a funzione.

Un puntatore a funzione è un normalissimo numero Long a 32 bit che fa riferimento ad una funzione in memoria. Il valore di tale numero corrisponde all'[indirizzo](#) della funzione stessa. In questo articolo svilupperemo una classe di nome **clsFunPtr** in grado di eseguire funzioni a partire da un puntatore alla stessa.

Saranno prodotti due differenti metodi  per l'assegnazione ed il recupero del puntatore alla funzione da eseguire. La prima di queste due soluzioni sfrutta un operatore di VB5: **AddressOf**; la seconda soluzione, utilizzando l'API, carica in memoria una libreria esterna e recupera il puntatore alla funzione della libreria tramite *GetProcAddress*.

Prima di approfondire la prima soluzione è fondamentale comprendere il funzionamento e le limitazioni nell'uso dell'operatore **AddressOf**. Si tratta di un operatore unario e non di una funzione; non può pertanto essere richiamato utilizzando le parentesi che delimitano gli argomenti nelle funzioni. La limitazione principale invece riguarda il recupero di Sub e Funzioni esclusivamente pubbliche e poste all'interno di un modulo  standard. Questo significa che non è possibile utilizzare AddressOf con funzioni private o metodi di classe  i quali comprendono funzioni e Sub all'interno di form, moduli di classe, controlli utente  e documenti [ActiveX](#)  in quanto tutti considerati delle classi a tutti gli effetti. Un'altra limitazione, ma facilmente superabile, riguarda l'assegnazione del valore restituito esclusivamente a funzioni. Questo comporta che non sarà possibile assegnare ad una variabile o scrivere all'interno di una proprietà l'indirizzo della funzione puntata. Il valore di AddressOf potrà essere assegnato esclusivamente ad argomenti di funzione.

In ogni caso non è possibile eseguire una funzione di cui si conosce l'indirizzo in memoria a meno di usare certe funzioni [API](#). L'operazione di esecuzione della funzione prende il nome di [Callback](#). La forma più comune per l'esecuzione di una funzione di Callback consiste nell'utilizzo della funzione API **CallWindowProc**.

Preso coscienza di queste limitazioni possiamo iniziare lo sviluppo della classe clsFunPtr che si presenta con le dichiarazioni dei valori membro e delle funzioni API:

```
1. Option Explicit
2.
3. Private lngPtrFunzione As Long
```

```

4. Private lngLibreria As Long
5.
6. Private Declare Function CallWindowProc0 Lib "user32" Alias
  "CallWindowProcA" (ByVal lpPrevWndFunc As Long) As Long
7. Private Declare Function CallWindowProc1 Lib "user32" Alias
  "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long) As Long
8. Private Declare Function CallWindowProc2 Lib "user32" Alias
  "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As
  Long) As Long
9. Private Declare Function CallWindowProc3 Lib "user32" Alias
  "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As
  Long, ByVal wParam As Long) As Long
10. Private Declare Function CallWindowProc4 Lib "user32" Alias
  "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As
  Long, ByVal wParam As Long, ByVal lParam As Long) As Long

```

Le due variabili alle righe 3 e 4 sono utilizzate rispettivamente per contenere il puntatore alla funzione da eseguire e l'[handle](#) alla [libreria](#) esterna caricata in memoria.

La funzione *CallWindowProc* dichiarata alle righe 6-10 con cinque differenti [Alias](#) consente di richiamare la funzione di Callback dato un puntatore e specificando gli eventuali argomenti a 32 bit per l'esecuzione. Naturalmente la prima di queste funzioni non richiede alcun argomento, mentre la quinta funzione ne richiede ben quattro. È stato necessario creare 5 differenti alias per dar modo di eseguire funzioni che accettino fino a quattro parametri in [input](#).

```

11. Private Declare Function LoadLibrary Lib "kernel32" Alias "LoadLibraryA" (ByVal
  lpLibFileName As String) As Long
12. Private Declare Function GetProcAddress Lib "kernel32" (ByVal hModule As Long,
  ByVal lpProcName As String) As Long
13. Private Declare Function FreeLibrary Lib "kernel32" (ByVal hLibModule As Long) As
  Long
14.

```


La funzione *LoadLibrary* consente di caricare in memoria una libreria esterna il cui nome è specificato con argomento *lpLibFileName*. Alla riga 12 invece è dichiarata la funzione *GetProcAddress* per il recupero di un puntatore a funzione di una libreria esterna. L'ultima funzione API è *FreeLibrary* che, come dice il nome, scarica la libreria caricata con *LoadLibrary* e ne libera la memoria.

È possibile assegnare alla classe il puntatore alla funzione di Callback in tre differenti maniere:

- Attraverso la proprietà **Funzione**
- Richiamando la Sub **AssegnaFunzione**
- Richiamando la funzione **FunzioneEsterna**

Prima di vedere in dettaglio le tre modalità ricordiamoci di una particolare limitazione accennata qualche riga addietro: non è possibile assegnare il valore restituito da *AddressOf* ad una variabile oppure ad una proprietà. Sembrerebbe quindi impossibile poter assegnare alla proprietà **Funzione** il puntatore. Ci viene in aiuto una semplicissima funzione [wrapper](#) di nome **FunPtr2LngPtr** che, in una maniera davvero semplice riceve come argomento di funzione il valore riportato da *AddressOf* e lo restituisce in uscita come valore di ritorno. Questo ci consente di assegnare alla proprietà **Funzione** non direttamente il risultato di *AddressOf* ma il risultato del richiamo di una funzione che come tale accetta l'uso di *AddressOf*. La nostra brevissima **FunPtr2LngPtr** si compone di una sola riga:

```
15. Public Function FunPtr2LngPtr(ByVal FunPtr As Long) As Long
16.     FunPtr2LngPtr = FunPtr
17. End Function
18.
```

La funzione di fatto riceve il valore **FunPtr** e lo restituisce in uscita. Sembrerebbe inutile ma ci consente di evitare quella sciocca limitazione di VB. Possiamo allora procedere con la visione della proprietà  **Funzione**, altrettanto banale:

```
19. Public Property Get Funzione() As Long
20.     Funzione = lngPtrFunzione
21. End Property
22.
23. Public Property Let Funzione(ByVal newFunzione As Long)
24.     lngPtrFunzione = newFunzione
25. End Property
26.
```

La proprietà di fatto assegna alla variabile [membro](#) **lngPtrFunzione** il valore che riceve.

```
27. Public Sub AssegnaFunzione(ByVal PtrFunzione As Long)
28.     lngPtrFunzione = PtrFunzione
29. End Sub
30.
```

La Sub **AssegnaFunzione** effettua la medesima operazione dell'assegnazione alla proprietà **Funzione**. Sfruttando questa routine è però possibile evitare l'uso della funzione wrapper vista in precedenza ed assegnare il valore riportato da AddressOf direttamente alla variabile membro.

```
31. Public Function FunzioneEsterna(ByVal strLibreria As String, ByVal strFunzione As
    String) As Long
32.     If lngLibreria <> 0 Then FreeLibrary lngLibreria
33.     lngLibreria = LoadLibrary(strLibreria)
34.     If lngLibreria <> 0 Then lngPtrFunzione = GetProcAddress(lngLibreria,
        strFunzione)
35.     FunzioneEsterna = lngPtrFunzione
36. End Function
37.
```

L'ultima funzione di assegnazione del puntatore a funzione è **FunzioneEsterna**. A differenza delle altre due non utilizza e non vuole un puntatore a funzione. Essa infatti richiede il passaggio di due stringhe **strLibreria** e **strFunzione** per la specifica della libreria esterna da caricare e della funzione di cui recuperare il puntatore.

La funzione si apre con un controllo d'obbligo: se la variabile **lngLibreria** contiene un valore diverso da zero sarà segno che è stata caricata una libreria esterna in precedenza e pertanto dovrà essere scaricata dalla memoria tramite *FreeLibrary* (riga 32).

Fatto questo sarà possibile caricare la nuova libreria (riga 33) ed assegnare l'[handle](#) del modulo caricato alla variabile **lngLibreria**. Nel caso che la libreria esterna non possa essere caricata o non fosse presente, la funzione *LoadLibrary* restituirà valore 0. Se quindi il valore di **lngLibreria** è differente da zero allora la libreria esterna potrà essere interrogata per consentire il recupero del puntatore alla funzione esterna tramite *GetProcAddress* (riga 34). Il valore restituito sarà quindi assegnato alla variabile membro **lngPtrFunzione**. Il valore di ritorno della funzione sarà lo stesso puntatore recuperato.

Non ci resta che l'ultima routine della classe: quella di esecuzione della funzione di cui è stato recuperato il puntatore. La funzione prende il nome di **Esegui**:

```

38. Public Function Esegui(ByVal intArgomenti As Integer, Optional ByVal lngArg1 As
    Long = &H0&, Optional ByVal lngArg2 As Long = &H0&, Optional ByVal lngArg3 As Long
    = &H0&, Optional ByVal lngArg4 As Long = &H0&) As Long
39.     Select Case intArgomenti
40.         Case 0: Esegui = CallWindowProc0(lngPtrFunzione)
41.         Case 1: Esegui = CallWindowProc1(lngPtrFunzione, lngArg1)
42.         Case 2: Esegui = CallWindowProc2(lngPtrFunzione, lngArg1, lngArg2)
43.         Case 3: Esegui = CallWindowProc3(lngPtrFunzione, lngArg1, lngArg2, lngArg3)
44.         Case Else: Esegui = CallWindowProc4(lngPtrFunzione, lngArg1, lngArg2,
        lngArg3, lngArg4)
45.     End Select
46. End Function
47.
```

Il primo argomento obbligatorio specifica quanti parametri dovranno essere forniti alla funzione da richiamare. In base al valore di *intArgomenti* sarà richiamato l'[alias](#) alla funzione *CallWindowProc* corrispondente. I quattro argomenti opzionali sono tutti interi a 32 bit ed assumono il valore predefinito di 0 nel caso che non venissero forniti.

La routine andrà a richiamare la funzione [API](#) *CallWindowProc* che si occuperà di eseguire la chiamata della funzione indicata da *lngPtrFunzione*.

### Attenzione!

Un uso scorretto della funzione *CallWindowProc* con il richiamo di routine esterne, nella maggioranza dei casi genera un errore irreversibile che termina il programma.

L'errore più comune è quello di fornire alla funzione uno o più valori errati.

Un altro errore molto comune è dato dall'errato numero di argomenti passati alla funzione. In tal caso la funzione API verrà comunque eseguita ma al ritorno al programma chiamante sarà generato un errore di **Convenzione di chiamata non valida**.

Prima di testare il funzionamento della classe definiamo un controllo d'obbligo: al termine dell'uso di un'istanza, se è stata caricata una libreria esterna, essa andrà scaricata tramite *FreeLibrary*:

```

48. Private Sub Class_Terminate()
49.     If lngLibreria <> 0 Then FreeLibrary lngLibreria
50. End Sub
```

Passiamo alla dimostrazione del funzionamento della classe **clsFunPtr**: ci servirà un form un modulo standard per contenere le funzioni pubbliche da puntare tramite *AddressOf*. Il modulo conterrà due sole funzioni a scopo dimostrativo di nome **Funzione1** e **Funzione2**:

```

1. Option Explicit
2.
3. Public Function Funzione1() As Long
4.     Funzione1 = MsgBox("Esecuzione di Funzione1" & vbNewLine & "Sono le ore " &
        Time$, vbOKOnly)
5. End Function
6.
7. Public Function Funzione2() As Long
8.     Funzione2 = MsgBox("Esecuzione di Funzione2" & vbNewLine & "Scegli SI o NO",
        vbYesNo + vbInformation, "AddressOf")
9. End Function
```

Entrambe le funzioni mostrano una semplicissima finestra con un messaggio. Soltanto la seconda consente di eseguire una scelta tra i due pulsanti Si o No.

Il form conterrà invece un'unica routine posta a gestione dell'evento ⚡ **Activate** del Form:

```
1. Option Explicit
2.
3. Private Sub Form_Activate()
4.     Dim FunPtr1 As clsFunPtr
5.     Dim str1 As String
6.     Dim str2 As String
7.     Dim lngStr1 As Long
8.     Dim lngStr2 As Long
9.
```

Le 5 variabili dichiarate rappresentano rispettivamente un'istanza della classe `clsFunPtr`, due stringhe a scopo dimostrativo utilizzate più avanti ed i relativi due puntatori alle stringhe stesse, da fornire alla funzione API *ShellAboutA*.

```
10.     Me.AutoRedraw = True
11.     Set FunPtr1 = New clsFunPtr
```

Alla riga 10 viene impostata la proprietà `AutoRedraw` del form a `True` per consentire la corretta visualizzazione dei messaggi stampati sulla sua superficie. Alla riga 11 è istanziata una copia della classe appena sviluppata.

```
12.     FunPtr1.Funzione = FunPtr1.FunPtr2LngPtr(AddressOf Funzione1)
13.     Me.Print FunPtr1.Esegui(0)
14.
```

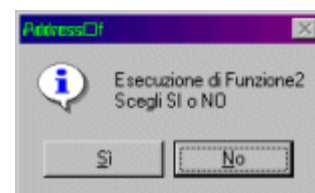
La prima funzione da richiamare è la **Funzione1** del modulo standard. Il riferimento è fatto usando la proprietà `Funzione` dell'istanza, accompagnata dal richiamo del [wrapper](#) `FunPtr2LngPtr` per l'assegnazione del valore restituito da `AddressOf`. Alla riga 13 è quindi eseguito il richiamo della funzione puntata. Poiché la funzione non richiede parametri aggiuntivi, sarà specificato il valore 0 come argomento per il metodo `Esegui`.



Il risultato della chiamata alla funzione mostrerà una finestra di avviso con il nome della funzione e l'orario attuale.

```
15.     FunPtr1.AssegnaFunzione AddressOf Funzione2
16.     Me.Print FunPtr1.Esegui(0)
17.
```

La seconda funzione verrà richiamata tramite l'uso del metodo **AssegnaFunzione**; in questo caso poiché il metodo è una routine non sarà necessario richiamare la funzione wrapper ma potremo passare l'indirizzo della funzione semplicemente utilizzando `AddressOf`. Anche questa seconda funzione non richiede parametri aggiuntivi e sarà quindi specificato il valore 0 ad indicare che non sono necessari argomenti ulteriori.



La **Funzione2** richiamata mostrerà una finestra di messaggio con la possibilità di scelta del valore Si o No, corrispondenti ai valori `vbYes` e `vbNo` dell'[enumerazione](#) `VbMsgBoxResult`.



In base al pulsante scelto sarà ritornato un valore e stampato sulla superficie del form, che vedremo al termine dell'articolo.

```
19.     str1 = StrConv("Puntatori a funzione#VB Simple", vbFromUnicode)
20.     str2 = StrConv("http://www.vbsimple.net", vbFromUnicode)
21.     lngStr1 = StrPtr(str1)
22.     lngStr2 = StrPtr(str2)
23.     Call FunPtr1.FunzioneEsterna("SHELL32.DLL", "ShellAboutA")
24.     Me.Print FunPtr1.Esegui(4, Me.hWnd, lngStr1, lngStr2)
25.
```

La terza funzione richiamata è leggermente più complessa delle altre: innanzitutto si tratta di una funzione esterna di nome *ShellAboutA* contenuta nella libreria *Shell32*. Richiede il passaggio di 4 argomenti:

- l'handle della finestra da cui è richiamata
- un puntatore ad una stringa in memoria da visualizzare nella parte superiore
- un puntatore ad una stringa in memoria da visualizzare nella parte centrale
- un valore 0 non trattato in questa sede

Alla riga 19 è costruita la prima stringa **str1**, che si compone di due parti (secondo le convenzioni della funzione *ShellAboutA*). La parte alla sinistra del simbolo # apparirà sulla barra del titolo della finestra di dialogo, mentre la parte alla destra apparirà nella parte superiore della finestra. La stringa alla riga 20, di nome **str2**, è il semplice indirizzo internet di VBSimple. Poiché entrambe le stringhe dovranno essere maneggiate tramite puntatore e la funzione *ShellAboutA* si aspetta il formato di testo [ANSI](#), sarà necessario convertire le due stringhe in questo formato tramite la funzione di VB **StrConv**.

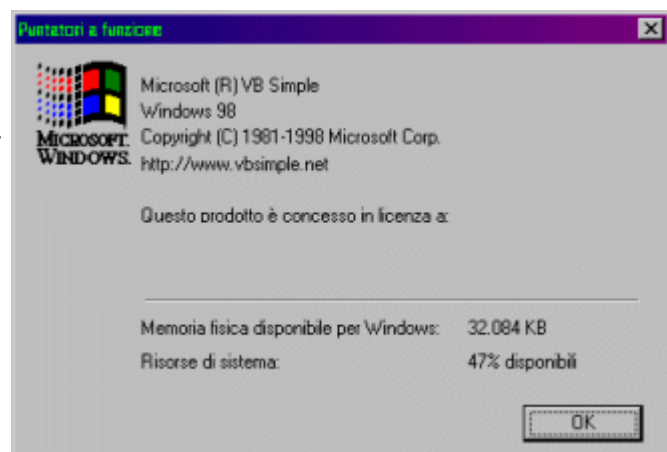
Alle righe 21 e 22 saranno recuperati i puntatori alle due stringhe tramite la funzione **StrPtr** e memorizzati nelle variabili **lngStr1** e **lngStr2**. Causa alcune particolari limitazioni di VB è stato necessario utilizzare le variabili di appoggio **str1** ed **str2**, altrimenti il contenuto delle stringhe sarebbe stato invalidato subito dopo il loro uso.

Finalmente alla riga 23 è recuperato il puntatore alla funzione tramite l'uso del metodo **FunzioneEsterna**, semplicemente specificando la libreria ed il nome della funzione da puntare. Così alla riga 24 detta funzione potrà essere richiamata nel solito modo; i parametri richiesti sono 4 e corrispondono all'handle del form, al puntatore lngStr1, al puntatore lngStr2 ed un valore assunto di 0, non specificato perché automaticamente fatto dalla nostra funzione **Esegui**.

Il risultato dell'operazione produce la finestra di dialogo mostrata a fianco.

Sulla barra del titolo è riportato **"Puntatori a funzione"**, seguito dal logo di Windows, con il corrispondente copyright. Accanto a questo è riportato il nome del programma (nel nostro caso **"VB Simple"**).

Qualche riga più giù troviamo l'indirizzo internet di VBSimple, preceduto e seguito da alcune notizie di copyright, licenza e



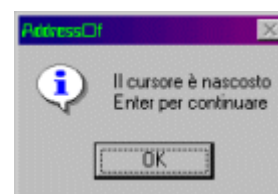
risorse disponibili.

```
26.      Call FunPtr1.FunzioneEsterna("KERNEL32.DLL", "GetCurrentProcessId")
27.      Me.Print FunPtr1.Esegui(0)
28.
```

Un altro esempio dell'uso del metodo **FunzioneEsterna** è dato dalla riga 26: verrà richiamata la funzione *GetCurrentProcessId* dalla libreria *Kernel32*, che restituisce in uscita l'identificativo del [processo](#) in esecuzione. Poiché la funzione non richiede parametri aggiuntivi sarà richiamato il metodo **Esegui** con il valore 0.

```
29.      Call FunPtr1.FunzioneEsterna("USER32.DLL", "ShowCursor")
30.      Me.Print FunPtr1.Esegui(1, 0)
31.      MsgBox "Il cursore è nascosto" & vbNewLine & "Enter per continuare",
vbInformation
32.      Me.Print FunPtr1.Esegui(1, 1)
33.      MsgBox "Il cursore è riapparso", vbInformation
```

L'ultima funzione esterna richiamata è *ShowCursor* nella libreria *User32*. La funzione richiede un parametro unico che indica se il cursore dovrà essere nascosto o mostrato. La stessa funzione è pertanto richiamata una prima volta con il parametro 0 ad indicare che il cursore dovrà essere celato, in seguito a cui sarà visualizzato un avviso informativo (righe 30 e 31).



Poiché il puntatore del mouse verrà nascosto, risulterà difficile premere il pulsante OK per proseguire. Basterà pertanto premere il tasto Enter per far svanire la finestra di avviso e continuare l'esecuzione del programma.

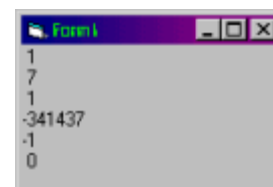
La stessa funzione, senza che sia necessario puntarla nuovamente tramite l'uso del metodo **FunzioneEsterna**, sarà eseguita una seconda volta, ma stavolta con il parametro 1 ad indicare che il cursore dovrà essere nuovamente mostrato.



Anche questa volta sarà mostrato un avviso informativo della ricomparsa del cursore precedentemente nascosto.

Tutti i valori restituiti dalle chiamate alle funzioni via puntatore saranno mostrati sulla superficie del form, a puro scopo dimostrativo:

- La prima riga mostra il valore 1 corrispondente a **vbOk** nella finestra di avviso della **Funzione1**
- La seconda riga mostra il valore 6 o 7 corrispondente alle costanti **vbYes** e **vbNo** nella finestra di avviso della **Funzione2**
- La terza riga indicherà il risultato del richiamo della finestra di dialogo *ShellAboutA*. Un valore di 0 indicherà la fallita visualizzazione mentre il valore 1 indicherà che l'operazione è riuscita con successo.
- Il valore della quarta riga riporterà l'identificativo del processo in corso
- Gli ultimi due valori sono i risultati della funzione *ShowCursor*.



La classe sviluppata si presenta molto semplice e riutilizzabile in qualunque progetto con

estrema semplicità. L'unico punto debole riguarda l'uso scorretto della funzione *CallWindowProc* fornendo valori errati per un funzione esterna. In tal caso è assolutamente normale aspettarsi un crash del programma, fino addirittura al blocco del sistema operativo ([BSOD](#)).

[Fibia FBI](#)

7 Agosto 2002

[Torna all'indice degli HowTo](#)

---