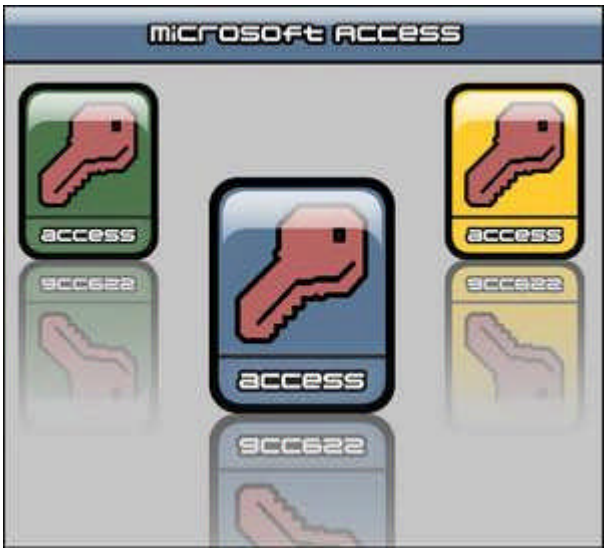
SUBCLASSING TUTORIAL



MS ACCESS

INDICE

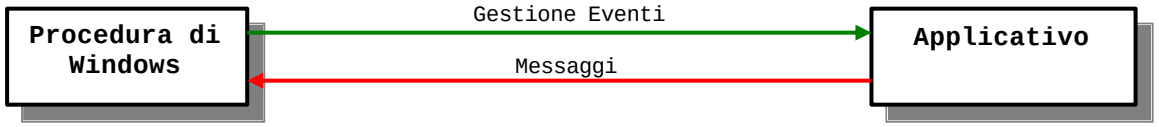
INTRODUZIONE

Quando viene premuto un tasto o un pulsante del mouse oppure quando una finestra di un'applicazione viene visualizzata, chiusa, spostata o ridimensionata, ... vengono generati eventi.

Un evento non è nient'altro che la risposta ad un'azione compiuta dall'utente, dal sistema o da un'applicazione (quindi come risultato di una procedura): ad esempio se premiamo un tasto su una casella di testo, viene generato l'evento KeyPress. Ogni oggetto risponde a determinati eventi ed il più delle volte la risposta ad un evento è programmabile: ad esempio posso far sì che sull'evento KeyPress dell'oggetto TextBox mi venga indicato il carattere premuto:

```
Private Sub TextBox_KeyPress(KeyAscii As Integer)
      MsgBox Chr(KeyAscii)
End Sub
```

Ma come fa la finestra che riceve la pressione del tasto a stabilire che evento deve generare? Quando viene premuto un tasto, viene generato un Messaggio associato alla finestra, che corrisponde all'azione intrapresa. Il messaggio viene inviato alla procedura di Windows(Standard) che lo elabora una procedura che chiameremo windowproc, ed in base a tale messaggio, quindi ai suoi parametri, sa quello che è successo (evento) e di conseguenza cosa deve fare (procedura associata all'evento).



Il messaggio in questione viene inviato con questi parametri:

```
hwnd, Msg, wparam, lparam
```

```
hwnd        L'Handle della finestra preposta a ricevere il messaggio
Msg         Identificativo del Messaggio o dell'Evento
wparam      Parametro aggiuntivo che prende significato a seconda di Msg
lparam      Parametro aggiuntivo che prende significato a seconda di Msg
```

Msg è anch'esso un numero che identifica univocamente un messaggio, mentre wParam ed lParam non sempre sono indispensabili, ma possono offrire informazioni utili a completare il messaggio.

Ad esempio, se muovo il Mouse nell'Area del Corpo della Form, verrà generato il messaggio WM_MOUSEMOVE la cui struttura sarà:

```
hwnd        Me.hWnd
Msg         WM_MOUSEMOVE = &H200
wparam      MK_LBUTTON = &H1 dettaglio sui VirtualKey(del Mouse) premuti
lparam      posizione del puntatore del mouse in coordinate X/Y
```

In dettaglio il messaggio di notifica viene strutturato come segue:

WM_MOUSEMOVE

```
WM_MOUSEMOVE

    WPARAM wParam
    LPARAM lParam;
```

Parametri

wParam
Indica qualsiasi dei VirtualKeys premuti. Questo parametro può essere uno o più combinazioni dei seguenti valori:

MK_CONTROL
    CTRL key è premuto.
MK_LBUTTON
    Il Pulsante Mouse_Sinistro è premuto.
MK_MBUTTON
    Il pulsante Mouse_Centrale è premuto.
MK_RBUTTON
    Il Pulsante Mouse_Destro è premuto.
MK_SHIFT
    SHIFT key è premuto.
MK_XBUTTON1
    Windows 2000/XP: The first X button is down.
MK_XBUTTON2
    Windows 2000/XP: The second X button is down.

```
'//Key State Masks for Mouse Messages
Public Const MK_LBUTTON = &H1
Public Const MK_RBUTTON = &H2
Public Const MK_SHIFT = &H4
Public Const MK_CONTROL = &H8
Public Const MK_MBUTTON = &H10
```

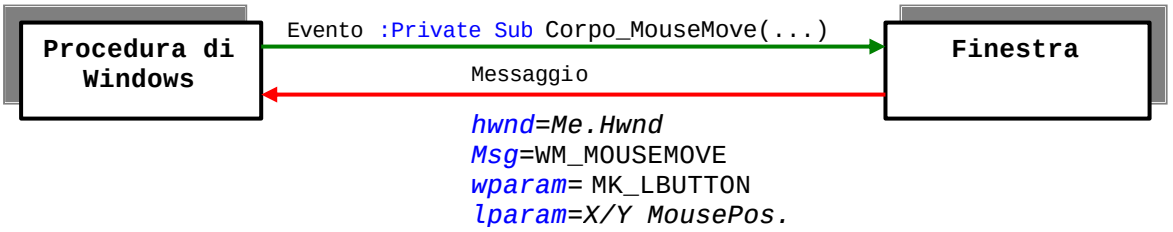
lParam
La parte bassa della parola specifica la Coordinata X del puntatore relativa all'angolo superiore sinistro dello schermo.
La parte alta della parola specifica la Coordinata Y del puntatore relativa all'angolo superiore sinistro dello schermo.

In questo caso, Access ha trasformato il secondo parametro 32-bit del messaggio (contenente la posizione di x e di y in pixel, a 16 bit ciascuno) in valore Single con unità di misura in twips, ed il primo parametro lo ha scomposto per restituire quali combinazioni di VirtualKeys erano presenti, ottenendo in questo modo la funzione Evento così esposta:

```
Private Sub Corpo_MouseMove( Button         As Integer, _
                             Shift          As Integer, _
                             X              As Single, _
                             Y              As Single)

      ‘ Azione
End Sub
```

Graficamente possiamo riassumere il passaggio in questo modo:



Ora, ad esempio voi avete necessità di avere le coordinate del mouse in pixel, Access li ha convertiti in twips ed ora voi le convertirete ancora in pixel. Oltre al tempo perso, è in qualche modo irritante sapere che Windows restituisce valori con un formato differente da quello di cui abitualmente si serve, sapendo che avrete bisogno di riconvertirli.

Generalizzando possiamo dire che ogni controllo può generare questo Evento:

```
Private Sub nomecontrollo_MouseMove(Button  As Integer, _
                                    Shift   As Integer, _
                                    X       As Single, _
                                    Y       As Single)
End Sub
```

I parametri che otteniamo dall'Evento non sono altro che l'interpretazione degli stessi ottenuti dal messaggio di Notifica analizzato in dettaglio sopra:

Argomento	Descrizione
nomecontrollo	Il nome del controllo a cui viene applicata la routine evento Mouse Spostato.
Button	Lo stato dei pulsanti del mouse quando si verifica l'evento. Se è necessario provare l'argomento Button, è possibile utilizzare una delle seguenti costanti intrinseche come maschere di bit: Costante — Descrizione **acLeftButton** — La maschera di bit per il pulsante sinistro del mouse. **acRightButton** — La maschera di bit per il pulsante destro del mouse. **acMiddleButton** — La maschera di bit per il pulsante centrale del mouse.
Shift	Lo stato dei tasti MAIUSC, CTRL e ALT quando il pulsante specificato dall'argomento Button è stato premuto o rilasciato. Se è necessario provare l'argomento Shift, è possibile utilizzare una delle seguenti costanti intrinseche come maschere di bit: Costante — Descrizione **acShiftMask** — La maschera di bit per MAIUSC. **acCtrlMask** — La maschera di bit per CTRL. **acAltMask** — La maschera di bit per ALT.
X, Y	Le coordinate x e y per la posizione corrente del puntatore del mouse. Gli argomenti X e Y sono sempre espressi in twips.

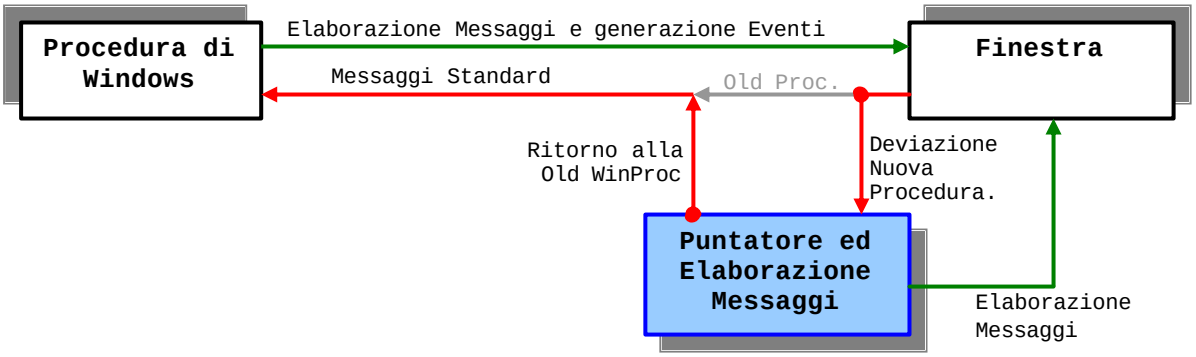
Ora ci chiederemo se è possibile ricevere questi messaggi direttamente, senza appoggiarsi all'interprete VB(che li converte in Eventi ma a volte con valori poco utili)...?

RISPOSTA

La risposta ovviamente è **SI**, la tecnica da usare è il **SubClassing**, ma dovreste usarla soltanto se realmente necessario in quanto è abbastanza rischiosa e può compromettere la stabilità delle vostre applicazioni nel caso non venga strutturata in modo adeguato.

SUBCLASSING

E' l'elaborazione dell'intercettazione dei messaggi di Windows che un programma in Visual Basic non riceverebbe normalmente.
In pratica quello che si fa è dire a Windows che i messaggi di Notifica relativi ad una certa Finestra devono essere deviati alla nostra nuova procedura prima di essere processati.



In questo modo possiamo mirare al messaggio che ci interessa, operare per eseguire una certa azione, quindi ripristinare il normale funzionamento.
Per dire a Windows di effettuare questa DEVIAZIONE serve usare la funzione **SetWindowLong** api per installare un nuovo puntatore dei messaggi alla nostra nuova WindowProc, quindi per ripristinare il lavoro di WIN dovremo ritornare al punto d'interruzione passandolo con la funzione **CallWindowProc**.
Il primo parametro da specificare come dicevamo è l'HANDLE(Hwnd) della finestra da SubClassare, il secondo è il messaggio GWL_WNDPROC (-4) che specifica a WINDOWS che vogliamo SubClassare, quindi il terzo è l'indirizzo della nostra nuova Procedura.
Questa procedura viene chiamata ogni volta che WINDOWS è attivo e qualcosa sta per essere processato, o qualche parametro di sistema viene modificato da altri processi.
Se **SetWindowLong** ritorna un valore=0 significa che si è verificato un errore, normalmente dovrebbe restituire l'indirizzo della vecchia procedura(quello che dovremo poi usare per far proseguire il nomale processamento)

SETWINDOWLONG

Vediamo nel dettaglio la struttura della nostra funzione API. Come vedremo questa funzione non serve solo per SubClassare, ma ha diverse potenzialità a seconda della combinazione dei parametri.

```
Public Declare Function SetWindowLong Lib "user32" _
               Alias "SetWindowLongA" _
               (ByVal hwnd                As Long, _
               ByVal nIndex               As Long, _
               ByVal dwNewLong            As Long) As Long
```

Parametri

hWnd
Identifica la Finestra e, indirettamente, la classe alla quale appartiene.

nIndex
Specifica l'offset(*Base Zero*) del valore da impostare come indice. Valore valido è qualsiasi valore da ZERO al numero di bytes della memoria extra della finestra meno 4; per impostare altri valori vedere quelli sotto previsti:

GWL_EXSTYLE
Imposta i nuovi stili tra quelli previsti nelle estensioni di stile.

Esempio:
SetWindowLong Me.hwnd, GWL_EXSTYLE, WS_EX_LAYERED

In questo esempio usato nel "**Demo Form_Ombra**" imposto lo stile della Finestra previsto nelle estensioni a *Layered*, in modo tale da potergli poi passare i parametri di trasparenza e opacità.

GWL_STYLE
Imposta i nuovi stili tra quelli Standard.

GWL_WNDPROC
Questa è l'opzione più importante per il nostro argomento.
Imposta il nuovo indirizzo per la *Procedura Personalizzata.*

GWL_HINSTANCE
Imposta un nuovo Handle per l'applicazione.

GWL_ID
Imposta un nuovo identificatore per la Finestra.

GWL_USERDATA
Sets the 32-bit value associated with the window. Each window has a corresponding 32-bit value intended for use by the application that created the window.

I seguenti valori sono disponibili anche quando l'Hwnd identifica il DialogBox:

DWL_DLGPROC
Imposta il nuovo indirizzo della procedura del DialogBox.

DWL_MSGRESULT
Imposta il valore di ritorno del messaggio processato dalla procedura DialogBox.

DWL_USER
Imposta informazioni personalizzate proprie dell'applicazione come Handle o Puntatori.

dwNewLong
Specifica il valore di sostituzione.

CALLWINDOWPROC

Questa funzione sarà importante per ripristinare il puntatore di procedura di Windows al termine del Subclassing.

```
Public Declare Function CallWindowProc 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
```

Parametri

lpPrevWndFunc
Puntatore alla precedente WindowsProc(quella sospesa o deviata).

hWnd
Identifica l'Handle della finestra che riceve la procedura.

Msg
Specifica il messaggio.

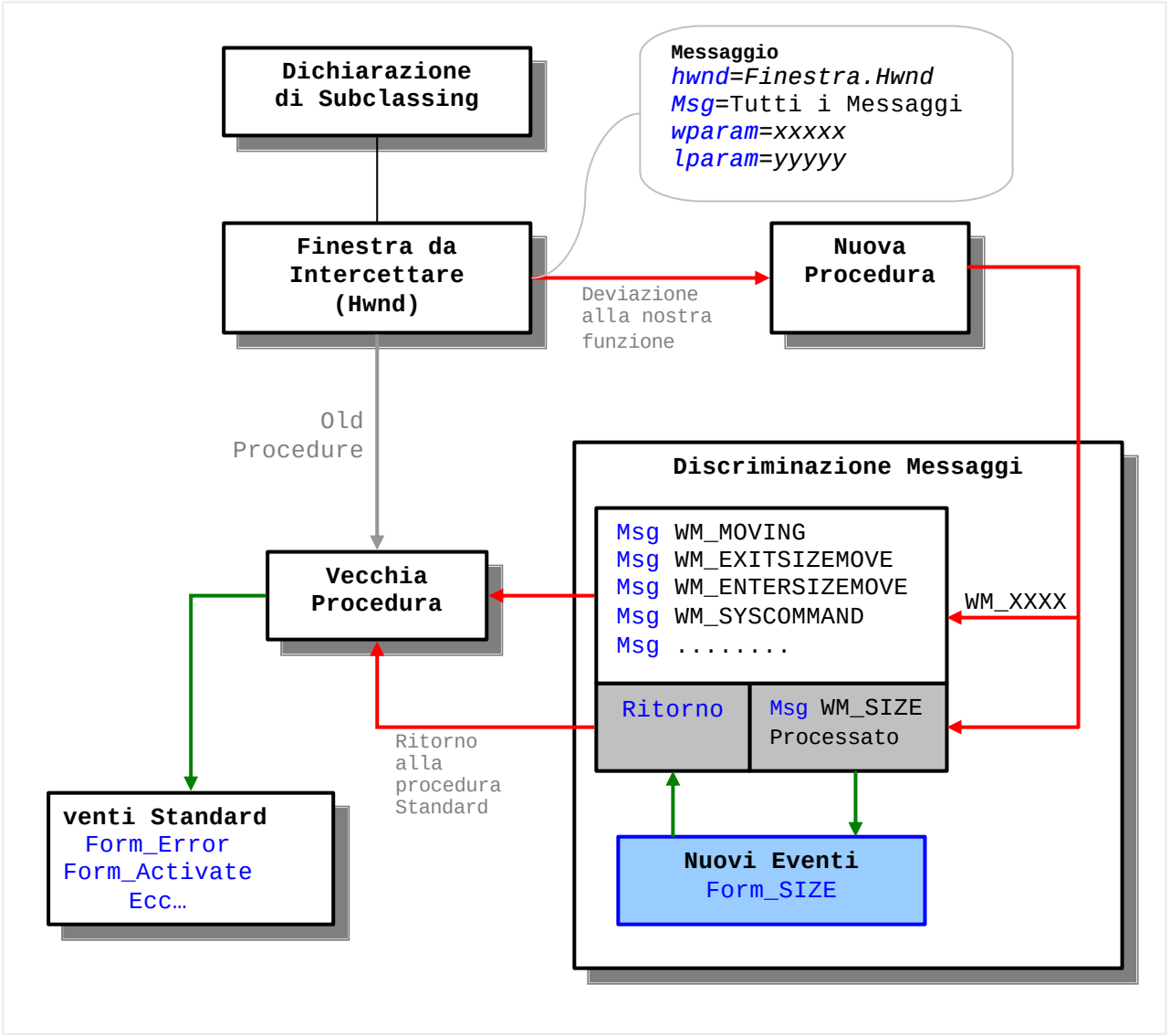
wParam
Informazioni addizionali dipendenti da Msg.

lParam
Informazioni addizionali dipendenti da Msg.

Quindi sull'Evento Load della Maschera definiremo che abbiamo intenzione di processare i Messaggi di una certa Finestra e dove deviarli per il processamento.

La corretta sequenza degli eventi è quella visualizzata nello schema sotto che per comodità analizza il caso della creazione dell'evento MOVE di Maschera:

ESEMPIO




```
Private Sub Form_Load()
     ' Attivazione del SubClassing sulla Form
     OldWindowProc = SetWindowLong(Me.hwnd, _
                                   GWL_WNDPROC, _
                                   AddressOf NewWindowProc)

End Sub

Private Sub Form_Unload(Cancel As Integer)
    SetWindowLong Me.hWnd, GWL_WNDPROC, lProcOld
End Sub

Per capire:
      Me.Hwnd          'è l'Handle dell'Oggetto da SubClassare
      GWL_WNDPROC      'indica a WIN che vogliamo SubClassare
      NewWindowProc    'è la nostra nuova Procedura che riceve i Messaggi

Ovviamente non dobbiamo dimenticare che alla chiusura della Finestra
subclassata serve ripristinare la procedura originale indicando a Windows che
```

non deve più essere deviata alla nostra procedura, altrimenti verrebbe generato un Crash.

Ora passiamo alla nostra Nuova Procedura:

```
Public Const GWL_WNDPROC  As Long = (-4)
' Imposta il nuovo indirizzo per la WindowProc

Public Function NewWindowProc(ByVal hwnd       As Long, _
                              ByVal msg        As Long, _
                              ByVal wParam     As Long, _
                              ByVal lParam     As Long) As Long

      Debug.Print   “Handle = “ & hwnd & _
                    “Messaggio = “ & msg & _
                    “wParam = “ & wParam & _
                    “lParam = “ & lParam

      ‘ Azioni dimostrative per l’intercettazione dei Messaggi:
      Select Case msg
            Case WM_MOVING
                  ‘ Azione 1 Messaggio elaborato
                  ‘ Deviazione alla Nostra Funzione
                  ‘ per il processamento.
            Case WM_GETMINMAXINFO
                  ‘ Azione 2 Messaggio elaborato
                  ‘ Deviazione alla Nostra Funzione
                  ‘ per il processamento.
            Case Else
                  ‘ Qui abbiamo l’elenco di tutti i Messaggi
                  ‘ che non dobbiamo elaborare
      End Select

      ' Continua il normale processamento. IMPORTANTISSIMO!
      NewWindowProc = CallWindowProc(OldWindowProc, _
                                     hwnd, msg, _
                                     wParam, lParam)
End Function
```

Questa tecnica è applicabile a Maschere, controlli o altri oggetti che hanno un Handle(hWnd) definibile.
Access gestisce l’Handle dei controlli in modo differente da VB come descritto nel paragrafo “**HANDLE DEI CONTROLLI**”, in fondo al tutorial, pertanto richiede più attenzione.
La nuova WindowProc deve essere l'indirizzo di una funzione definita in un modulo BAS.

Ora quando l'oggetto riceve un messaggio da Windows, il sistema chiama la nuova WindowProc la quale esamina il messaggio ed intraprende l’azione adatta in base all’esigenza.
Se il messaggio ricevuto non è uno di quelli che ci interessano, la nostra WindowProc dovrà passare il riferimento alla funzione originale di elaborazione dei messaggi **CIÒ È ESTREMAMENTE IMPORTANTE !**
Se non passate i messaggi alla funzione originale, l'oggetto non potrà aggiornarsi ovvero non verranno effettuate le azioni di ripristino estetico(PAINT) e funzionale, e non verranno effettuate le altre funzioni standard di Windows dando origine al Crash.

VARIABILI STRUTTURATE

Molti messaggi restituiscono nei parametri *lParam* e *wParam* un pointer di tipo *Long* alla locazione di memoria che contiene una struttura dati.
I dati contenuti nella struttura puntata dal Parametro sono spesso molto importanti e devono essere recuperati per potervi accedere.
Sotto abbiamo 2 esempi di messaggi di Notifica che restituiscono *lParam* come pointer a strutture dati complesse.

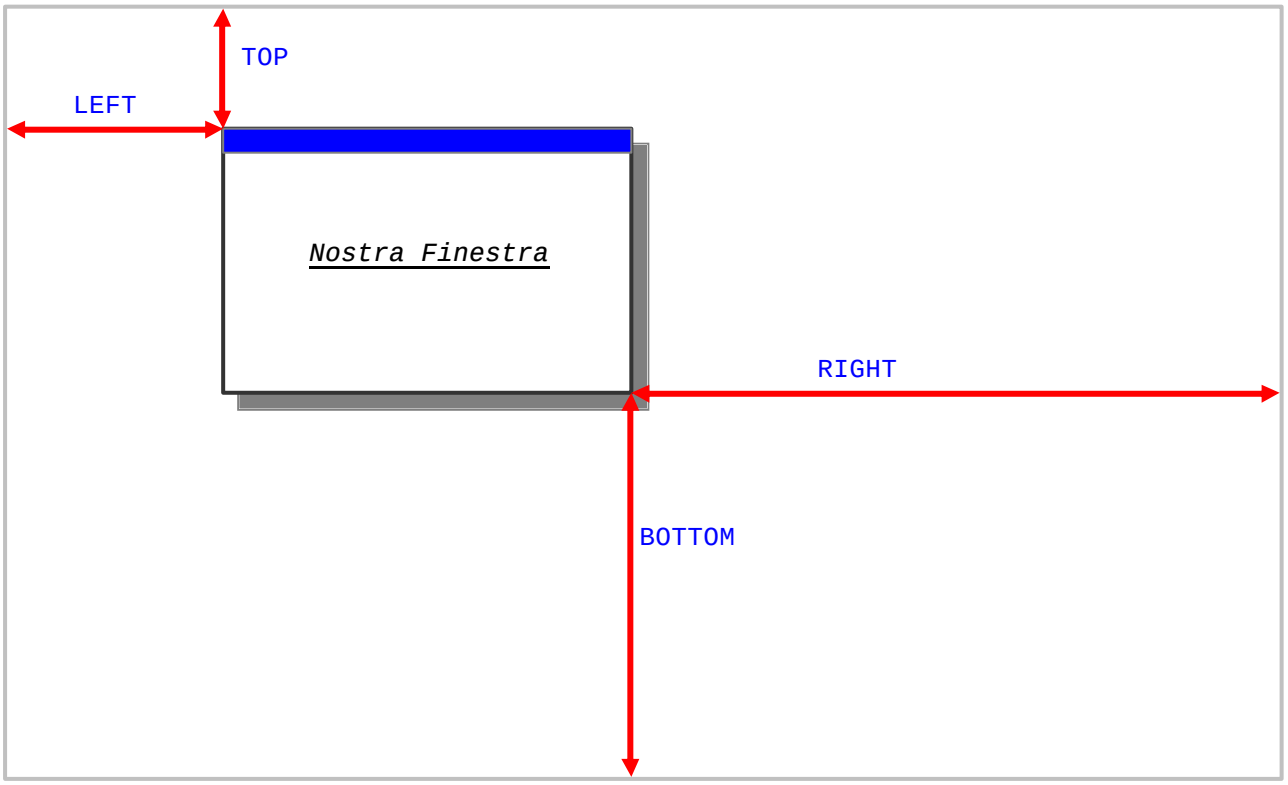
ESEMPIO 1 RECT

```
WM_MOVING
    WPARAM          wParam
    LPARAM          lParam;
```

La variabile lParam è un pointer ad una struttura di tipo *RECT*.
La variabile RECT rappresenta le coordinate rettangolari relative all'oggetto, tali coordinate sono memorizzate nei membri della struttura detti anche Atomi.

Dichiarazione:

```
Public Type RECT
     Left                As Long
     Top                 As Long
     Right               As Long
     Bottom              As Long
End Type
```



ESEMPIO 2 *WINDOWPOS*

La struttura dati *WINDOWPOS* può contenere informazioni riguardanti le dimensioni e la posizione della nostra Form.

Dichiarazione:

```
Type WINDOWPOS
     hWnd                As Long
     hWndInsertAfter     As Long
     x                   As Long
     y                   As Long
     cx                  As Long
     cy                  As Long
     Flags               As Long
End Type
```

Membri o Atomi della struttura

hwnd
Handle della finestra.

hwndInsertAfter
Specifica la posizione della finestra in ordine Z(Fronte-Retro nel senso della profondità di visualizzazione).
Questo membro può essere anche l'Handle della finestra dietro la quale la nostra Form è piazzata, oppure può essere un valore speciale compreso nella lista della funzione *SetWindowPos*

x
Posizione LEFT della finestra.
y
Posizione TOP della finestra.
cx
Larghezza della finestra, in pixels.
cy
Altezza della finestra, in pixels.

flags
Specifica la posizione della Finestra. I membri di "**flags**" possono essere uno o più dei seguenti valori.

SWP_DRAWFRAME
Disegna una struttura (definita nella descrizione della Classe della finestra) intorno alla finestra.

SWP_FRAMECHANGED
Trasmette un messaggio di WM_NCCALCSIZE alla finestra, anche se il formato della finestra non sta cambiando. Se questo Flag non è specificato il messaggio di notifica citato viene inviato soltanto quando il formato della finestra sta cambiando.

SWP_HIDEWINDOW
Nasconde la finestra.

SWP_NOACTIVATE
Non attiva la finestra. Se questo Flag non è settato, la finestra viene attivata e portata in primo piano o immediatamente dietro la finestra che ha come Handle quello specificato in hwndInsertAfter.

SWP_NOCOPYBITS

Scarta l'intero contenuto della zona client. Se questo Flag non è specificato, il contenuto valido della client-area è conservato e copiato nuovamente dentro client-area dopo che la finestra sia graduata o riposizionata.

SWP_NOMOVE
Mantiene la posizione corrente ignorando i parametri **x** e **y**.

SWP_ NOOWNERZORDER
Blocca la priorità della finestra nell'ordine Z(Profondità).

SWP_NOREDRAW
Non ridisegna i cambiamenti. Se Flag è settato, non viene generato il REPAINT. Ciò si applica alla Client-Area, alla nonClient-area (barra compresa di titolo e barre di scorrimento) ed a qualsiasi parte della finestra Parent scoperta come conseguenza della finestra che è spostata.
Se il Flag è settato, l'applicazione deve invalidare o ridisegnare esplicitamente tutte le parti della finestra Parent che necessitano di essere ridisegnate.

SWP_NOREPOSITION
Stesso di SWP_NOOWNERZORDER.

SWP_NOSENDCHANGING
Annulla il CallBack della finestra impedendo che riceva il messaggio WM_WINDOWPOSCHANGED

SWP_NOSIZE
Mantiene le dimensioni attuali ignorando i parametri **cx** e **cy**.

SWP_NOZORDER
Mantiene la priorità attuale in Z ignorando il parametro **hwndInsertAfter**.

SWP_SHOWWINDOW
Mostra la finestra.

Valori delle Costanti dei Messaggi di Notifica

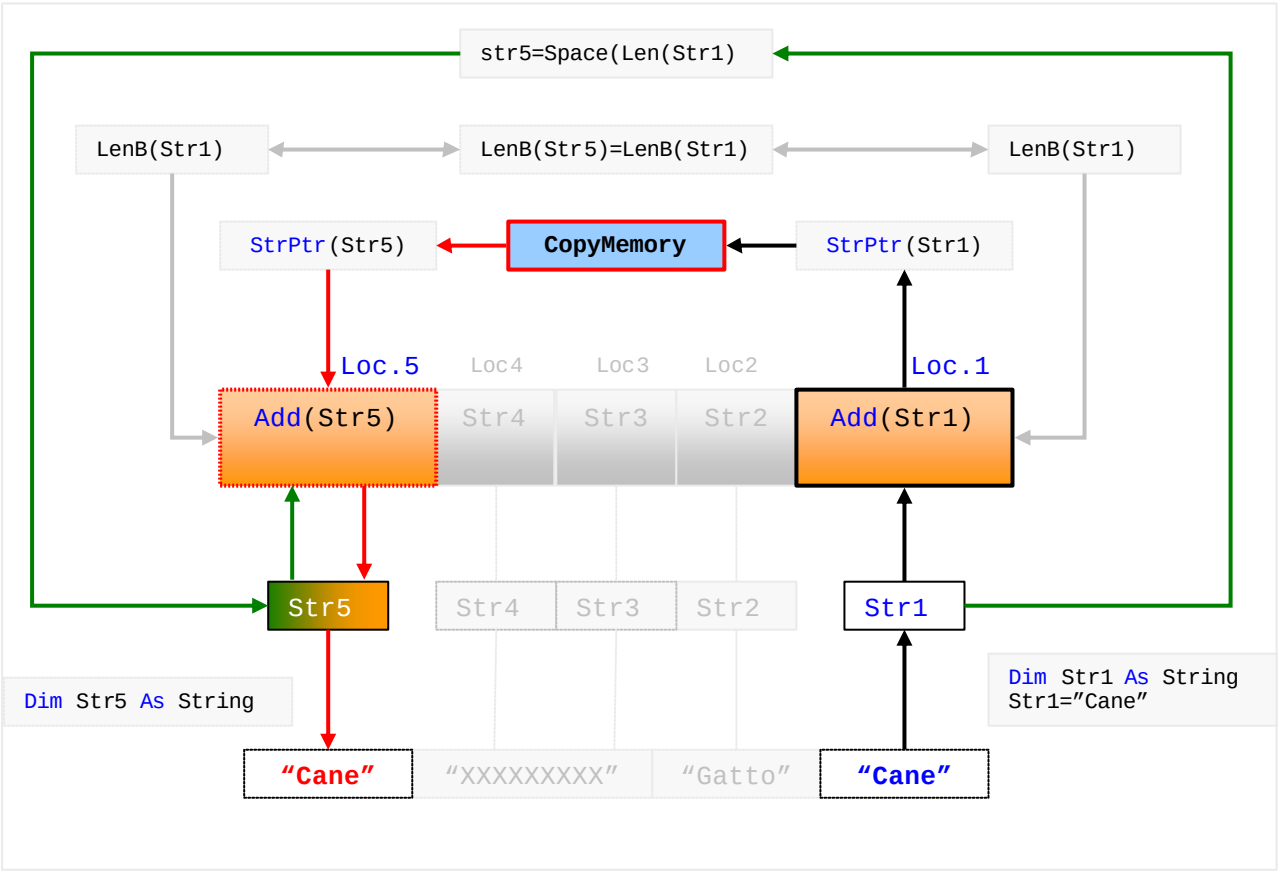
```
' ---------------- DICHIARAZIONE A LIVELLO DI MODULO ----------------
' SetWindowPos Flags
Public Const SWP_NOSIZE = &H1
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOZORDER = &H4
Public Const SWP_NOREDRAW = &H8
Public Const SWP_NOACTIVATE = &H10
Public Const SWP_FRAMECHANGED = &H20
'  Il cambio del Frame invia la notifica WM_NCCALCSIZE
Public Const SWP_SHOWWINDOW = &H40
Public Const SWP_HIDEWINDOW = &H80
Public Const SWP_NOCOPYBITS = &H100
Public Const SWP_NOOWNERZORDER = &H200
'  Don't do owner Z ordering
Public Const SWP_DRAWFRAME = SWP_FRAMECHANGED
Public Const SWP_NOREPOSITION = SWP_NOOWNERZORDER
' --------------------------------------------------------------------
```

Per recuperare i dati contenuti nella locazione indicata dobbiamo copiare l'area di memoria in una struttura dati equivalente, pertanto dovremo dichiarare una variabile strutturata dello stesso tipo di quella puntata, quindi copiarci dentro il contenuto.
Per eseguire questa operazione abbiamo bisogno di una funzione che possa copiare aree di memoria in modo corretto: l'API *COPYMEMORY*.

TUTORIAL COPYMEMORY

Questa funzione è estremamente importante, in quanto i messaggi di notifica come abbiamo visto e come vedremo, possono restituire nella variabile *lParam/wParam* un puntatore ad un area di memoria popolata da una struttura dati.
Al fine di poter accedere ai valori della struttura dobbiamo ovviamente poterli copiare in una struttura equivalente dichiarata nel nostro progetto.

Copia un blocco di Memoria da una locazione all'altra. Nel demo che propongo non faremo altro che copiare una Variabile stringa in un'altra, ma sfruttando la loro locazione di memoria. Questo processo è estremamente più veloce.



Se seguiamo questo schema con quanto esposto nella sequenza commentata del codice, riusciremo a renderci conto del flusso logico legato all'uso di questa funzione, ed a capire il motivo per cui sia estremamente veloce dal momento che non muove Bytes ma solo puntatori di memoria.

ESEMPIO 1

```
Private Declare Sub CopyMemory Lib "kernel32" _
                        Alias "RtlMoveMemory" _
                        (pDest                 As Any, _
                        pSource                As Any, _
                        ByVal ByteLen          As Long)

Private Sub Form_Load()

    Dim MyStr As String
    MyStr = "Il Cane è bello!"

    'La variabile MyStr è in memoria ed ora dobbiamo recuperare l'indirizzo
    'nella quale risiede. Per questo usiamo la funzione StrPtr, che
    'restituisce un Pointer di locazione alla variabile stringa:

    Dim MyStrAddr As Long
    'Ora ricaviamo il Puntatore
    MyStrAddr = StrPtr(MyStr)

    'Per poter copiare un'area in una seconda area dobbiamo dichiararla e
    'ovviamente impegnarne una dimensione sufficiente a contenerne quello che
    'deve ricevere.
    'Nel nostro caso deve ricevere il contenuto della stringa MyStr, pertanto
    'dobbiamo allocare l'equivalente area in memoria.

    Dim MyNewStr As String
    'Allochiamo un'area di memoria UGUALE alla nostra MyStr
    MyNewStr = Space(Len(MyStr))

    'Quindi dobbiamo estrarre il Pointer alla nostra variabile MyNewStr.
    Dim MyNewStrAddr As Long
    MyNewStrAddr = StrPtr(MyNewStr)

    'In questo caso ByVal è estremamente importante poiché la chiamata a
    'CopyMemory non sa che intenzioni abbiamo relativamente alle variabili
    'che vengono passate.
    'L'indirizzo del Pointer va passato ByVal(a valore) e non
    'ByRef(riferimento).
    'LenB è differente da Len poiché restituisce il numero di Bytes e non la
    'lunghezza della stringa.
    'Quindi Copiamo la quantità di Bytes della MyStr allocati in MyStrAddr
    'nella locazione MyNewStrAddr:
    CopyMemory ByVal MyNewStrAddr, ByVal MyStrAddr, LenB(MyStr)

    'Ora verifichiamo che in MyStrAddr ci sia "Il Cane è bello!"
    MsgBox MyNewStr
End Sub
```

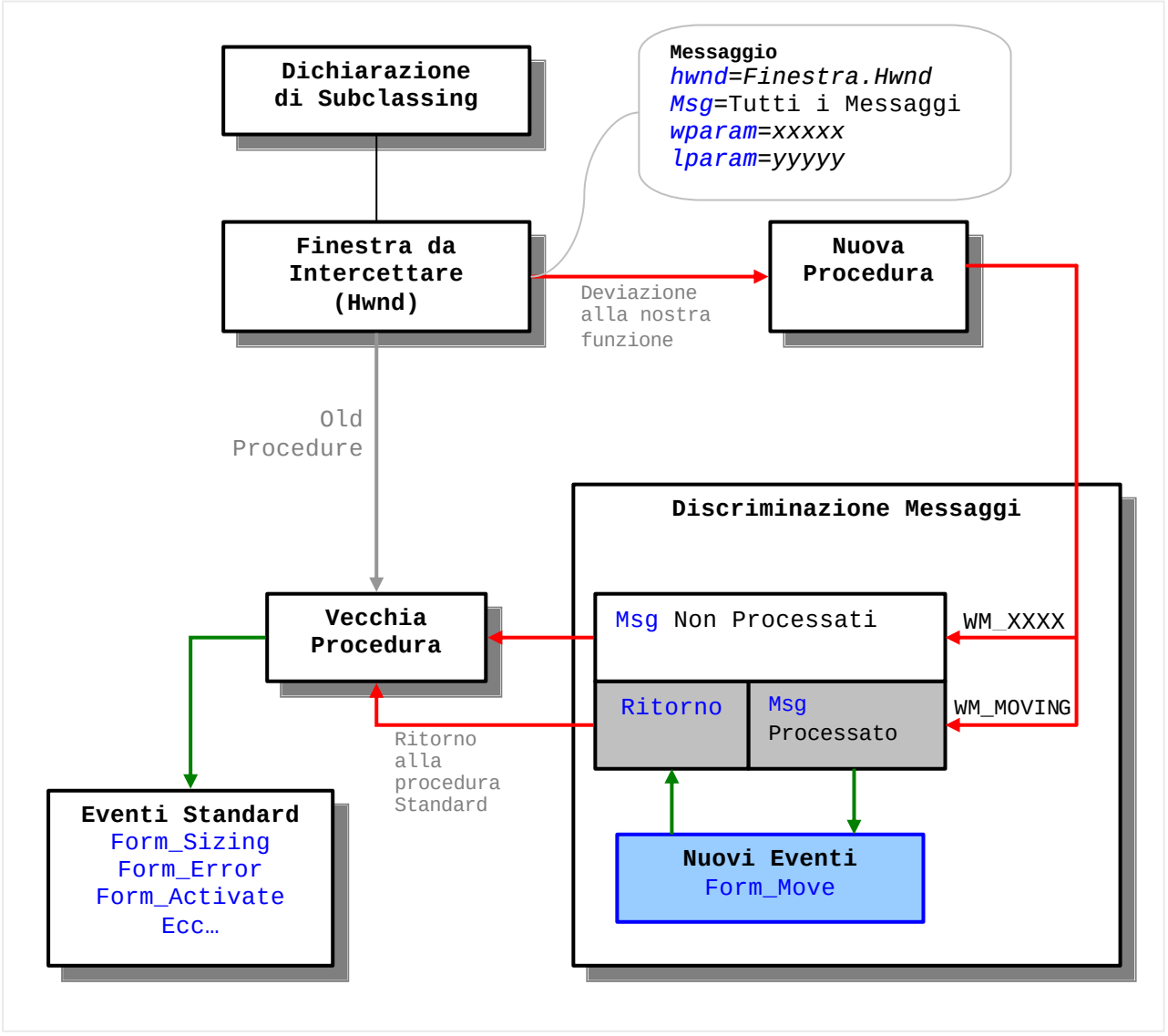
ESEMPIO 2

In pratica abbiamo fatto una cosa equivalente a questa, ma in memoria:

```
Private Sub Form_Load()
    Dim MyStr As String, MyNewStr As String
    MyStr = "Il Cane è bello!"
    MyNewStr = MyStr
    MsgBox MyNewStr
End Sub
```

ESEMPIO FORM_MOVE(EVENT)

L'esempio più classico dell'esigenza dell'utilizzo del SubClassing è quello di intercettare l'Evento Form_Move non presente in modo nativo negli eventi della Classe Form al fine di poterlo processare.
Per questo nella nostra procedura **NewWindowProc** dovremo preoccuparci di intercettare il messaggio **WM_MOVING** = &H216 chiamando la nostra Funzione.



WM_MOVING

E' inviato alla finestra che l'utente sta muovendo. Per processare questo messaggio, un'applicazione può monitorare la posizione del rettangolo della finestra e, se serve, modificarla.

Sintassi

```
WM_MOVING

    WPARAM wParam
    LPARAM lParam;
```

Parametri

wParam
Non usato.

lParam
Puntatore alla locazione che restituisce una struttura di tipo ***RECT*** contenente le attuali coordinate della finestra, in coordinate di schermo(pixels). Per modificare la posizione del rettangolo(quindi della nostra finestra) dobbiamo cambiare i membri o atomi della struttura. Come abbiamo visto l'unico modo per accedere ai dati puntati da lParam è copiarli in una variabile(strutturata) equivalente.

Valore restituito
TRUE se il messaggio viene processato.

Visto che intercettando questo messaggio riceviamo un parametro(lParam) che rappresenta un Puntatore ad una struttura, ricordiamo che, per renderla fruibile, ovvero per poter accedere ai sui dati dobbiamo appoggiarci all'API *CopyMemory* la quale consente di ricopiare l'area di memoria puntata in quella della nostra variabile strutturata:

CODICE

```
' ---------------- DICHIARAZIONE A LIVELLO DI FORMS ----------------
Private Sub Form_Load()
    ' Attivazione del SubClassing sulla Form
    OldWindowProc = SetWindowLong(Me.hwnd, _
                                  GWL_WNDPROC, _
                                  AddressOf NewWindowProc)

End Sub

Private Sub Form_Unload(Cancel As Integer)
    ‘ Disattivazione del Puntatore alla nostra Procedura
    SetWindowLong Me.hWnd, GWL_WNDPROC, OldWindowProc
End Sub

‘ Nuovo EVENTO MOVE di Maschera
Public Sub Form_Move(X As Long, Y As Long, CX As Long, CY As Long)
    Debug.Print “La Maschera si Muove”
    Me.Caption= “X= ”  & X   & “ “ & _
                “Y= ”  & Y   & “ “ & _
                “CX= ” & CX  & “ “ & _
                “CY= ” & CY
End Sub
```

Il messaggio da intercettare come dicevamo sarà WM_MOVING = &H216, pertanto la WindowProc sarà così definita:

```
' ---------------- DICHIARAZIONE A LIVELLO DI MODULO ----------------
Public Declare Function CallWindowProc 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

Public Declare Function SetWindowLong Lib "user32" _
            Alias "SetWindowLongA" _
            (ByVal hwnd             As Long, _
            ByVal nIndex            As Long, _
            ByVal dwNewLong         As Long) As Long

Public Declare Sub CopyMemory Lib "kernel32" _
            Alias "RtlMoveMemory" _
            (pDest                  As Any, _
            pSource                 As Any, _
            ByVal ByteLen           As Long)

Type RECT
    Left                            As Long
    Top                             As Long
    Right                           As Long
    Bottom                          As Long
End Type

Public Const GWL_WNDPROC  As Long = (-4)
' Imposta il nuovo indirizzo per la WindowProc
Public OldWindowProc          As Long

Public Function NewWindowProc(ByVal hwnd       As Long, _
                              ByVal msg        As Long, _
                              ByVal wParam     As Long, _
                              ByVal lParam     As Long) As Long

      Const WM_MOVING = &H216
      Dim   rct              As Rect

      Select Case msg
            Case WM_ MOVING
                  CopyMemory rct, ByVal lParam, LenB(rct)
                  ‘ Quì chiamiamo il Metodo Form_Move della nostra
                  ‘ Form che abbiamo reso opportunamente Public.
                  With rct
                      Forms!NomeForm.Form_Move(.Left,.Top,.Right,.Bottom)
                  End With
            Case Else
      End Select
      ‘ E’ estremamente importante ripristinare la chiamata originale
      WindowProc = CallWindowProc(OldWindowProc, hWnd, Msg, wParam, lParam)
End Function
```

La funzione deve essere definita in un modulo standard per poter soddisfare i requisiti dell'operatore AddressOf. Quindi la nuova Window Procedure verifica se il messaggio ricevuto sia WM_MOVING e lo dirotta verso una nostra nuova azione emulando l’Evento Form_Move.

PERCHÈ SUBCLASSARE

Usando questa tecnica possiamo completamente sostituire o implementare le funzione standard di Windows con una nostra, di conseguenza possiamo rispondere ai messaggi stessi in un modo che Visual Basic non può fare, noi possiamo decidere se trasmettere i messaggi alle funzioni originali o no, se modificarli prima di restituirli nel modo più adatto alle nostre esigenze.

Questo ci consente di manipolare un'estrema quantità di Eventi e condizioni non nativi del linguaggio di programmazione.

La tecnica in questione però non è priva di rischi, e questo è dovuto in particolare all'interazione con uno strato di programmazione al quale noi non accediamo, e che si aspetta comportamenti prestabiliti.
Di conseguenza se prendiamo questa strada non dobbiamo commettere errori, in quanto le procedure di Windows manderebbero in Crash il sistema.
C'è poi da specificare che l'uso di questa tecnica in Access non è molto convenzionale, per diversi motivi:

1) Quello che personalmente ritengo più pericoloso e problematico è che Access non consente di far girare in IDE il SubClassing, questo richiede SEMPRE di chiudere il progetto e riaprirlo senza accedere al codice per testare l'effetto.
   La soluzione più semplice a questo problema è quella di affidare ad una DLL esterna la gestione del SubClassing(intesa solo come intercettazione dei messaggi che poi verrebbero inviati alla nostra applicazione dove devono essere Processati).

2) L'unica classe che possiede l'Handle identificabile è la Classe Form, tutte le altre Classi(TextBox, ListBox, CommandButton) sono viste come semplici immagini finché non ricevono lo stato attivo(Focus), a quel punto ereditano completamente le proprietà della classe, questo in Access rende snella l'esecuzione, ma rappresenta un limite nel nostro caso.

Qui sotto esponiamo alcuni DEMO semplificati per comprendere come interpretare le istruzioni di MSDN relative ai messaggi di Notifica.

ESEMPIO 1 (Ricavare le Coordinate e le Dimensioni dopo la variazione)

Quello che vogliamo ottenere è che spostando o ridimensionando la nostra Form venga aggiornata la caption della stessa con i dati ricavati dalla nostra azione.
E' chiaro che in questo caso avremo potuto inserire altre azioni, come magari vincolare la posizione di una seconda Form alla nostra, ottenendo così l'effetto di ANCORAGGIO.

Purtroppo non esiste in Access l'Evento Form_Move, ma solo l'Evento Form_Resize, il problema è che per questi scopi non è consono usarlo in quanto verrebbero generati sfarfallii dell'immagine dovuti al repaint.

Quindi al fine di controllare ed intervenire su questi *EVENTI* sfruttiamo la tecnica in questione intercettando il messaggio WM_WINDOWPOSCHANGED.
Per semplicità ci limiteremo ad estrarre i valori di Posizione e Dimensione della nostra Form che il messaggio in questione ritorna nella struttura dati WINDOWPOS attraverso la variabile lParam, quindi dovremo capire come poter popolare una variabile di tipo strutturata con i dati contenuti dalla variabile in questione.

Visto che intercettando questo messaggio riceviamo un parametro(lParam) che rappresenta un Puntatore ad una struttura, occorre specificare che, per renderela fruibile dobbiamo appoggiarci all'API CopyMemory la quale consente di ricopiare l'area di memoria puntata nella nostra variabile strutturata:

```
Public Declare Sub CopyMemory Lib "kernel32" _
                   Alias "RtlMoveMemory" _
                   (pDest              As Any, _
                   pSource             As Any, _
                   ByVal ByteLen       As Long)
```

*In questo caso copia nella variabile **Info** un blocco di Memoria della dimensione dei Bytes richiesti dalla variabile **WINDOWPOS** puntato dall'indirizzo **Addr***

```
Dim Info As WINDOWPOS
Call CopyMemory (Info, ByVal Addr, lenB(Info))
```

Ora in Info avremo gli Atomi o Membri popolati con il contenuto corretto.

Notare l'uso del ByVal.

C'è la possibilità di dover modificare i valori per poi restituirli alla procedura, quindi serve sempre appoggiarsi alla funzione CopyMemory per effettuare il passaggio inverso:

```
Call CopyMemory (ByVal Addr, Info, LenB(Info))
```

N.B.:
Al fine di capire cosa stiamo facendo analizziamo perchè abbiamo scelto il Messaggio citato e di conseguenza occorre capire la funzione della variabile strutturata.

WM_WINDOWPOSCHANGED

Questo messaggio è inviato alla finestra la cui posizione/dimensioni o priorità sono cambiate in seguito all'intervento di un messaggio inviato tramite SetWindowPos o un'altra funzione di amministrazione, sostanzialmente potremo identificarlo in un Evento del tipo ***AfterPosSizeChange.***
La nostra finestra riceve il messaggio attraverso una WindowProc.

Sintassi

```
WM_WINDOWPOSCHANGED

    WPARAM wParam
    LPARAM lParam;
```

Parametri

wParam
Non usato.

lParam
Punta alla struttura ***WINDOWPOS*** che contiene le informazioni riguardanti le nuove dimensioni della finestra e la sua nuova posizione.
Per comprendere meglio questo parametro serve studiare la struttura dati indicata che analizzeremo sotto.

Valore restituito

Se un'applicazione processa questo messaggio, l'utilizzo tramite il SendMessage ritorna ZERO.
Ricordiamo che usando SendMessage possiamo forzare l'invio di un messaggio ad una finestra, e di conseguenza eseguire il processamento.

CODICE

Il codice è ben commentato per rendere più comprensibile lo svolgimento.

```
' ---------------- DICHIARAZIONE A LIVELLO DI MODULO ----------------
Public Declare Function CallWindowProc 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

Public Declare Function SetWindowLong Lib "user32" _
            Alias "SetWindowLongA" _
            (ByVal hwnd             As Long, _
            ByVal nIndex            As Long, _
            ByVal dwNewLong         As Long) As Long

Public Const GWL_WNDPROC  As Long = (-4)
' Imposta il nuovo indirizzo per la WindowProc
Public OldWindowProc          As Long

Public Type WINDOWPOS
    hwnd                      As Long
    hWndInsertAfter           As Long
    x                         As Long
    y                         As Long
    cx                        As Long
    cy                        As Long
    Flags                     As Long
End Type
```

```
Public Function NewWindowProc(ByVal hwnd       As Long, _
                              ByVal msg        As Long, _
                              ByVal wParam     As Long, _
                              ByVal lParam     As Long) As Long

      Const WM_WINDOWPOSCHANGED = &H47
      Dim retVal              As Long
      Dim mWNDPOS             As WINDOWPOS

      Select Case msg
            Case WM_WINDOWPOSCHANGED
               ‘ Eseguiamo preventivamente la procedura normale
               ‘ se la nostra azione non deve modificarne i parametri
               ‘ al fine di evitare flare video.
               retVal=CallWindowProc(OldWindowProc, _
                                     hwnd, msg, _
                                     wParam, lParam)

               ‘ Ora dobbiamo convertire la variabile lParam nella variabile
               ‘ strutturata di tipo WINDOWPOS dichiarata sopra [mWNDPOS]
               ‘ La conversione si effettua copiando l’area di memoria in cui
               ‘ risiede la variabile lParam nella variabile strutturata.
               CopyMemory mWNDPOS, ByVal lParam, LenB(mWNDPOS)
               WITH mWNDPOS
                    Me.Caption= “X=” & .X & “ “ & _
                                “Y=” & .Y & “ “ & _
                                “CX=” & .CX & “ “ & _
                                “CY=” & .CY
               END WITH

            Case Else
               retVal=CallWindowProc(OldWindowProc, _
                                     hwnd, msg, _
                                     wParam, lParam)
      End Select
      ' Continua il normale processamento. IMPORTANTISSIMO!
      NewWindowProc = retVal
End Function

' ----------------- DICHIARAZIONE A LIVELLO DI FORM -----------------
Private Sub Form_Load()
    ' Attivazione del SubClassing sulla Form
    OldWindowProc = SetWindowLong(Me.hwnd, _
                                  GWL_WNDPROC, _
                                  AddressOf NewWindowProc)
End Sub

Private Sub Form_Unload()
    ' Reset del SubClassing sulla Form
    SetWindowLong(Me.hwnd, _
                  GWL_WNDPROC, _
                  OldWindowProc)

End Sub
' ---------------------------------------------------------------------
```

ESEMPIO 2 (Intercettare le azioni del Mouse)

Visto che la tecnica è sempre la medesima, proponiamo lo studio di un nuovo Messaggio di Notifica WM_MOUSEWHEEL per l'intercettazione degli eventi del Mouse.

WM_MOUSEWHEEL

Questo messaggio è inviato alla finestra attiva quando la rotella del Mouse viene ruotata.
La procedura predefinita propaga il messaggio alla finestra Parent finchè non trova una finestra che non processa il messaggio.

La finestra lo riceve attraverso la sua funzione WindowProc.

Sintassi

```
WM_MOUSEWHEEL

    WPARAM wParam
    LPARAM lParam;
```

Parametri

wParam

La parte alta della parola indica l'angolo di rotazione espresso in multipli o divisioni di WHEEL_DELTA, che sono 120.
Un valore positivo indica che la rotella è stata girata avanti(lato opposto all'utente), negativo indica che la rotella è stata ruotata indietro, verso l'utente.
La parte bassa della parola indica se sono stati premuti più VirtualKeys.
Questo parametro può essere uno o più dei seguenti valori:

MK_CONTROL
CTRL key è premuto.
MK_LBUTTON
Il Pulsante Mouse_Sinistro è premuto.
MK_MBUTTON
Il pulsante Mouse_Centrale è premuto.
MK_RBUTTON
Il Pulsante Mouse_Destro è premuto.
MK_SHIFT
SHIFT key è premuto.
MK_XBUTTON1
Windows 2000/XP: The first X button is down.
MK_XBUTTON2
Windows 2000/XP: The second X button is down.

```
fwKeys = GET_KEYSTATE_WPARAM(wParam)          ;Restituisce un Integer
zDelta = GET_WHEEL_DELTA_WPARAM(wParam)       ;Restituisce un Byte
```

Queste Funzioni pare non siano disponibili da VisualBasic, quindi per estrarre il dato si ricorre all'estrazione della parte Alta e Bassa della Word:

```
fwKeys = (wParam AND &HFFFF&)
zDelta = (wParam AND &HFFFF0000) \ &H10000
```

lParam

La parte bassa della parola specifica la Coordinata X del puntatore relativa all'angolo superiore sinistro dello schermo.

La parte alta della parola specifica la Coordinata Y del puntatore relativa all'angolo superiore sinistro dello schermo.

```
xPos = GET_X_LPARAM(lParam)                    ;Restituisce un Integer
yPos = GET_Y_LPARAM(lParam)                    ;Restituisce un Integer
```

Queste funzioni sono disponibili in VB e VBA ma devono essere ricostruite :

Funzioni per ricavare le Coordinate X,Y del Mouse:

```
Public Function GET_X_LPARAM(ByVal lParam As Long) As Long
   Dim hexstr As String
   hexstr = Right("00000000" & Hex(lParam), 8)
   GET_X_LPARAM = CLng("&H" & Right(hexstr, 4))
End Function

Public Function GET_Y_LPARAM(ByVal lParam As Long) As Long
   Dim hexstr As String
   hexstr = Right("00000000" & Hex(lParam), 8)
   GET_Y_LPARAM = CLng("&H" & Left(hexstr, 4))
End Function
```

Valore Restituito
Se un'applicazione processa questo messaggio, l'utilizzo tramite il SendMessage ritorna ZERO.

CODICE
Ripropongo solo la WindowProc per la deviazione o disabilitazione della notifica del movimento della Rotella:

```
Public Function NewWindowProc(ByVal hwnd       As Long, _
                              ByVal msg        As Long, _
                              ByVal wParam     As Long, _
                              ByVal lParam     As Long) As Long

      Const WM_MOUSEWHEEL = &H20A
      Dim fwKeys             As long
      Dim zDelta             As long
      Select Case msg

            Case WM_MOUSEWHEEL
                  fwKeys = (wParam AND &HFFFF&)
                  zDelta = (wParam AND &HFFFF0000) \ &H10000
                  Forms!NomeForm.Caption= "VKeys= " & fwkeys & " " & _
                                          "Delta= " & zDelta
                  Exit Function     'Escludiamo l'effetto della Rotella

      End Select
      ' Continua il normale processamento. IMPORTANTISSIMO!
      NewWindowProc = CallWindowProc(OldWindowProc, _
                                     hwnd, msg, _
                                     wParam, lParam)

End Function
```

Eseguendo questa procedura potremo disabilitare l'effetto della Rotella del Mouse ed evitarne pertanto le conseguenze di spostamento RECORDS.
Questa procedura tuttavia non ha effetto sui controlli sottesi alla Form, come ListBox o TextBox(con ScrollBar).
Al fine di ottenere un funzionamento completo ***immagino****(non ho mai testato la cosa)* serva subclassare tutti i controlli in questione in grado di essere influenzati dallo SCROLL della rotella.

ESEMPIO 3 (Limitare le dimensioni MIN/MAX di una Form)

Con questo demo ci prefiggiamo di poter imporre alla nostra Finestra dei limiti di ridimensionamento sia MINIMI che MASSIMI, sostanzialmente faremo in modo che l'Evento di RESIZE ottenuto per trascinamento del bordi venga controllato al fine di imporre dei limiti.

WM_GETMINMAXINFO

Questo messaggio è inviato alla finestra attiva quando stiamo variando la posizione o la dimensione della stessa.
Un'applicazione può usare questo messaggio per riscrivere i valori di Default relativi al Massimo e Minimo ridimensionamento e posizione.

La finestra lo riceve attraverso la sua funzione WindowProc.

Sintassi

```
WM_GETMINMAXINFO

    WPARAM wParam
    LPARAM lParam;
```

Parametri

wParam
Non usato.

lParam
Punta alla struttura MINMAXINFO che contiene i valori di Default relativi alle massime *Posizioni* e *Dimensioni*, e i valori di Default minimi e massimi di *Ridimensionamento* per trascinamento del Bordo.
Un'applicazione può riscrivere questi valori settando i nuovi valori nei membri della struttura.

Valore Restituito
Se un'applicazione processa questo messaggio, l'utilizzo tramite il SendMessage ritorna ZERO.

MINMAXINFO

```
Type MINMAXINFO
    ptReserved          As POINTAPI
    ptMaxSize           As POINTAPI
    ptMaxPosition       As POINTAPI
    ptMinTrackSize      As POINTAPI
    ptMaxTrackSize      As POINTAPI
End Type
```

Membri o Atomi della struttura

ptReserved
Riservato, non utilizzato.

ptMaxSize
Specifica la larghezza massima (**POINT**. **x**) e l'altezza massima (**POINT**. **y**) della finestra. Per i sistemi a monitor Multiplo questo si riferisce al monitor principale.

ptMaxPosition
Specifica la massima posizione di Sinistra (**POINT**. **x**) e la massima posizione in alto (**POINT**. **y**).

Per i sistemi a monitor Multiplo questo si riferisce al monitor nel quale la finestra è massimizzata.

ptMinTrackSize
Specifica la larghezza Minima (**POINT**. **x**) e l'altezza minima (**POINT**. **y**) della finestra ottenibile per trascinamento del bordo. Rimane immutata per i sistemi con i video multipli.

ptMaxTrackSize
Specifica la larghezza Massima (**POINT**. **x**) e l'altezza Massima (**POINT**. **y**) della finestra ottenibile per trascinamento del bordo. Per i sistemi con i video multipli, questo è il formato per una finestra che è resa grande quanto lo schermo virtuale.

POINTAPI

```
Type POINTAPI
     x            As Long
     y            As Long
End Type
```

CODICE

```
Public Function NewWindowProc(ByVal hwnd       As Long, _
                              ByVal msg        As Long, _
                              ByVal wParam     As Long, _
                              ByVal lParam     As Long) As Long

  Dim mmiT As MINMAXINFO
  Const WM_GETMINMAXINFO = &H24
  Select Case msg
      Case WM_GETMINMAXINFO
          ' Copia il parametro nella variabile strutturata
          CopyMemory mmiT, ByVal lParam, LenB(mmiT)
          ' Nuove misure Height e Width MINIME di ridimensionamento
          mmiT.ptMinTrackSize.x = newXMIN
          mmiT.ptMinTrackSize.y = newYMIN
          ' Nuove misure di Height e Width MASSIME di ridimensionamento
          mmiT.ptMaxTrackSize.x = newXMAX
          mmiT.ptMaxTrackSize.y = newYMAX
          ' dopo aver modificato i valori in modo da vincolarli a quelli
          ' MIN/MAX dobbiamo ripassare la variabile a Windows, per
          ' ingannarlo, quindi copieremo la variabile nel Puntatore
          ' originale(lParam)
          CopyMemory ByVal lParam, mmiT, LenB(mmiT)

       Case Else
          NewWindowProc = CallWindowProc(OldWindowProc, _
                                         hwnd, msg, _
                                         wParam, lParam)
  End Select
End Function
```

Questo Demo dimostra come intercettare il messaggio di Resize e di alterarne l'esito ottenendo in questo modo la limitazione dell'azione stessa entro i nuovi limiti di MIN/MAX, la nostra Form non potrà allargarsi oltre il limite o restringersi sotto il limite sia sull'asse Verticale(Y) che Orizzontale(X).

ALTRI ESEMPI WINDOW MESSAGES

WM_ACTIVATE

Questo messaggio viene inviato ad sia alle finestre attive che a quelle non attive. Possiamo determinare cosa è successo alla nostra Finestra analizzando la parte bassa LOWORD del parametro lParam, mentre il parametro wParam restituisce l'handle della Finestra.

```
Public Function NewWindowProc(ByVal hwnd       As Long, _
                              ByVal msg        As Long, _
                              ByVal wParam     As Long, _
                              ByVal lParam     As Long) As Long

    Select Case Msg
        Case WM_ACTIVATE
            Select Case LOWORD(lParam)
                Case WA_ACTIVE
                    'Activated by some method other than a mouse click
                Case WA_CLICKACTIVE
                    'Activated by a mouse click
                Case WA_INACTIVE
                    'Deactivated
            End Select
    End Select

    NewWindowProc = CallWindowProc(lpfnOld, hWnd, uMsg, wParam, lParam)
End Function
```

WM_ACTIVATEAPP

Questo messaggio viene mandato quando la Finestra appartiene ad una differente applicazione che la finestra attiva sta per attivare. Il wParam è True se la nostra Finestra viene attivata o false se è un'altra applicazione ad essere attivata.

```
Public Function NewWindowProc(ByVal hwnd       As Long, _
                              ByVal msg        As Long, _
                              ByVal wParam     As Long, _
                              ByVal lParam     As Long) As Long

    Select Case Msg
        Case WM_ACTIVATEAPP
            Dim Active As Boolean
            Active = wParam
            If Active = True Then
                'La nostra Finestra è ATTIVA
            Else
                'E' stata attivata un'altra finestra
            End If
    End Select

    NewWindowProc = CallWindowProc(lpfnOld, hWnd, uMsg, wParam, lParam)
End Function
```

WM_CHILDACTIVATE

Questo messaggio viene inviato alla Finestra figlio quando l'utente seleziona la Caption(o titolo della Finestra) oppure quando la finestra viene attivata, mossa o ridimensionata. Questo messaggio non usa nessuno dei suoi parametri.

```
Public Function NewWindowProc(ByVal hwnd       As Long, _
                               ByVal msg        As Long, _
                               ByVal wParam     As Long, _
                               ByVal lParam     As Long) As Long

    Select Case Msg
        Case WM_CHILDACTIVATE
            'La finestra child è ha ricevuto il FOCUS.
    End Select
    NewWindowProc = CallWindowProc(lpfnOld, hWnd, uMsg, wParam, lParam)

End Function
```

WM_WINDOWPOSCHANGING

Questo messaggio viene inviato dalla finestra la cui Posizione/Dimensione e/o priorità sull'asse Z stanno per essere variate.
I valori restituiti nel parametro lParam(struttura *WINDOWPOS*) sono relativi alle nuove coordinate di rappresentazione, pertanto abbiamo la possibilità di ingannare Windows nel caso servisse.

```
Public Function NewWindowProc(ByVal hwnd       As Long, _
                               ByVal msg        As Long, _
                               ByVal wParam     As Long, _
                               ByVal lParam     As Long) As Long

      Const WM_WINDOWPOSCHANGING = &H46
      Dim mWNDPOS             As WINDOWPOS
      Dim retVal              As Long

      Select Case msg
            Case WM_ WINDOWPOSCHANGING
               ‘ Ora dobbiamo convertire la variabile lParam nella variabile
               ‘ strutturata di tipo WINDOWPOS dichiarata sopra [mWNDPOS]
               ‘ La conversione si effettua copiando l’area di memoria in cui
               ‘ risiede la variabile lParam nella variabile strutturata.
               CopyMemory mWNDPOS, ByVal lParam, LenB(mWNDPOS)
               ‘ Se volessi impedire il movimento potrei forzare quì la
               ‘ posizione passando dei parametri fissi alla variabile
               ‘ strutturata mWNDPOS
               WITH mWNDPOS
                    .X = myX
                    .Y = myY
                    .CX = myCX
                    .CY = myCY
               END WITH
               CopyMemory ByVal lParam, mWNDPOS, LenB(mWNDPOS)

     End Select

     NewWindowProc = CallWindowProc(lpfnOld, hWnd, uMsg, wParam, lParam)

End Function
```

WM_ENABLE

Questo messaggio è inviato quando un'applicazione cambia lo stato della finestra.

```
Public Function NewWindowProc(ByVal hwnd       As Long, _
                              ByVal msg        As Long, _
                              ByVal wParam     As Long, _
                              ByVal lParam     As Long) As Long

    Select Case Msg
        Case WM_ENABLE
            Dim Enabled As Boolean
            Enabled = wParam
            If Enabled = True Then
                'La nostra Finestra è Abilitata
            Else
                'La nostra Finestra è Disabilitata
            End If

    End Select
    NewWindowProc = CallWindowProc(lpfnOld, hWnd, uMsg, wParam, lParam)
End Function
```

WM_ENTERSIZEMOVE

Questo messaggio è inviato prima dell'inizio di una azione di Resize o di Move, i suoi parametri non espongono nulla, quindi è solo un messaggio d'Evento.

```
Public Function NewWindowProc(ByVal hwnd       As Long, _
                              ByVal msg        As Long, _
                              ByVal wParam     As Long, _
                              ByVal lParam     As Long) As Long

    Select Case Msg
        Case WM_ENTERSIZEMOVE
        'La nostra Finestra ha terminato l'azione o Evento di Moving o
        'Sizing appena l'utente ha rilasciato la Titlebar o il Bordo

    End Select
    NewWindowProc = CallWindowProc(lpfnOld, hWnd, uMsg, wParam, lParam)
End Function
```

WM_EXITSIZEMOVE

Questo messaggio è inviato dopo il termine di una azione di Resize o di Move. I suoi parametri non espongono nulla, quindi è solo un messaggio d'Evento.

```
Public Function NewWindowProc(ByVal hwnd       As Long, _
                              ByVal msg        As Long, _
                              ByVal wParam     As Long, _
                              ByVal lParam     As Long) As Long

    Select Case Msg
        Case WM_EXITSIZEMOVE
        'La nostra Finestra ha terminato l'azione o Evento di Moving o
        'Sizing appena l'utente ha rilasciato la Titlebar o il Bordo

    End Select
    NewWindowProc = CallWindowProc(lpfnOld, hWnd, uMsg, wParam, lParam)
End Function
```

WM_*SIZE, WM_SIZING, WM_SYSCOMMAND*

Questi messaggi sono tutti mirati alla gestione degli eventi di Resize della finestra analizzata. In realtà WM_SYSCOMMAND è un pò più complesso, ma nel nostro caso ci limitaiamo ad utilizzarlo per lo scopo in questione, vale a dire che intercetteremo *wParam* per discriminare il tentativo di esecuzione dell'Evento destinato al ridimensionamento una sorta di Before_Sizing.

WM_SIZE

Corrisponde ad una sorta di After_Size e per questo sfrutteremo sempre il parametro *wParam*. Questo messaggio è comodo per sapere l'azione che è stata apportata alla Finestra in questione, non è possibile annullarlo in quanto l'evento è accaduto.

Sintassi

```
WM_SIZE

    WPARAM wParam
    LPARAM lParam;
```

Parametri

wParam

Specifica il tipo di ridimensionamento richiesto, e può assumere uno dei seguenti valori.

SIZE_MAXHIDE

Inviato a tutte le finestre Popup quando qualche altra finestra viene MASSIMIZZATA.

SIZE_MAXIMIZED

La finestra è stata MASSIMIZZATA.

SIZE_MAXSHOW

Inviato a tutte le finestre Popup quando qualche altra finestra viene RIPRISTINATA alla sua dimensione originale(RESTORED).

SIZE_MINIMIZED

La finestra è stata MINIMIZZATA.

SIZE_RESTORED

La finestra è stata ridimensionata, ma né SIZE_MINIMIZED né SIZE_MAXIMIZED sono stati inviati.

lParam

La parte bassa della parola specifica la nuova larghezza della client area.
La parte alta della parola specifica la nuova Altezza della client area.

Valore restituito

Se un'applicazione elabora questo messaggio deve restituire ZERO.

WM_SIZING

Viene inviato quando l'utente sta effettuando il ridimensionamento. Come leggeremo in seguito in *lParam* viene restituito la variabile di tipo RECT che espone come membri le nuove dimensioni del rettangolo occupato dalla nostra Finestra, per bloccarle(inibirle) o per forzarle potremo pertanto accedere alla struttura modificandone il valore degli atomi.

Sintassi

```
WM_SIZING

    WPARAM wParam
    LPARAM lParam;
```

Parametri

wParam

Specifica in quale margine è effettuato il Resize. Può assumere uno dei seguenti valori.

WMSZ_BOTTOM
Margine inferiore

WMSZ_BOTTOMLEFT
Angolo in basso a sinistra

WMSZ_BOTTOMRIGHT
Angolo in basso a destra

WMSZ_LEFT
Margine sinistro

WMSZ_RIGHT
Margine destro

WMSZ_TOP
margine superiore

WMSZ_TOPLEFT
Angolo superiore sinistro

WMSZ_TOPRIGHT
Angolo superiore destro

lParam

Puntatore alla struttura *RECT* contenente le coordinate di schermo(pixels).
Per modificare la Dimensione o la Posizione del rettangolo occorre modificare i membri della struttura.

Valore restituito

L'applicazione che elabora questo messaggio deve restituire TRUEA.

WM_SYSCOMMAND

La finestra riceve questi messaggi quando l'utente sceglie un comando dal Menù contestuale di Windows oppure quando preme uno dei Pulsanti Maximize, Minimize, Restore o Close presenti nella Caption. Queste sono richieste inviate alla nostra Finestra, possono pertanto essere annullate impedendo che vangano eseguite. Corrispondono un pò ad un Evento Before_Event(Cancel as Integer) nel quale abbiamo la possibilità tramite il Cancel di annullare l'evento stesso. Vedremo in seguito come fare.

Sintassi

```
WM_SYSCOMMAND

    WPARAM wParam
    LPARAM lParam;
```

Parametri

wParam

Specifica il tipo di comando di sistema richiesto. Ecco alcuni dei valori che può assumere:

SC_CLOSE
Richiesta di CHIUSURA
SC_MAXIMIZE
Richiesta di MAXIMIZE
SC_MINIMIZE
Richiesta di MINIMIZE
SC_MOVE
Richiesta di MOVE
SC_RESTORE
Richiesta di RESTORE della posizione e delle dimensioni
SC_SIZE
Richiesta di RESIZE della finestra

Sono presenti altri valori attribuibili ma per non complicarci la cosa possono essere studiati su MSDN

lParam

E' un valore a 32Bit che viene scomposto in 2 valori a 16Bit i quali restituiscono le coordinate X/Y del MousePointer.

Valore restituito

L'applicazione che elabora questo messaggio deve restituire ZERO.

Codice

```
Public Function NewWindowProc(ByVal hwnd       As Long, _
                              ByVal msg        As Long, _
                              ByVal wParam     As Long, _
                              ByVal lParam     As Long) As Long

    Const WM_SIZE = &H5
    Const SIZE_RESTORED = 0
    Const SIZE_MINIMIZED = 1
    Const SIZE_MAXIMIZED = 2
    Const SIZE_MAXSHOW = 3
    Const SIZE_MAXHIDE = 4

    Const WM_SIZING = &H214

    Const WM_SYSCOMMAND = &H112
    'Const SC_SIZE = &HF000&       Per Access questo non è valido
    'Const SC_MOVE = &HF010&       Per Access questo non è valido
    Const SC_MOVE = &HF012&
    Const SC_MINIMIZE = &HF020&
    Const SC_MAXIMIZE = &HF030&
    Const SC_CLOSE = &HF060&
    Const SC_RESTORE = &HF120&

    Dim retVal   As Long

    Select Case msg
        Case WM_SYSCOMMAND
            Select Case wParam
            ‘ Possiamo generare gli EVENTI BEFORE_[wParam]
            ‘ ed eventualmente non farli procedere
                Case SC_MAXIMIZE
                    ‘ Se non voglio permettere l’azione di MAXIMIZE
                    ‘ intervengo e non la invio alla procedura standard.
                    If Not MaximizeIsAllowed Then Exit Function
                    retVal  =  CallWindowProc(lpfnOld,  hWnd,  uMsg,  wParam,
                    lParam)
                Case SC_MINIMIZE
                    ‘ Se non voglio permettere l’azione di MINIMIZE
                    ‘ intervengo e non la invio alla procedura standard.
                    If Not MinimizeIsAllowed Then Exit Function
                    retVal  =  CallWindowProc(lpfnOld,  hWnd,  uMsg,  wParam,
                    lParam)
                Case SC_RESTORE
                    ‘ Se non voglio permettere l’azione di RESTORE
                    ‘ intervengo e non la invio alla procedura standard.
                    If Not RestoreIsAllowed Then Exit Function
                    retVal  =  CallWindowProc(lpfnOld,  hWnd,  uMsg,  wParam,
                    lParam)
                Case SC_MOVE
                    ‘ Se non voglio permettere l’azione di MOVE
                    ‘ intervengo e non la invio alla procedura standard.
                    If Not MoveIsAllowed Then Exit Function
                    retVal  =  CallWindowProc(lpfnOld,  hWnd,  uMsg,  wParam,
                    lParam)
                Case Else
                    retVal  =  CallWindowProc(lpfnOld,  hWnd,  uMsg,  wParam,
                    lParam)
        End Select
```

```
            Case WM_SIZE
                Select Case wParam
                    ' Possiamo generare gli EVENTI AFTER_[wParam]
                    Case SIZE_MAXIMIZED
                    ' L'azione è avvenuta possiamo notificarla
                    Case SIZE_RESTORED
                    ' L'azione è avvenuta possiamo notificarla
                    Case SIZE_MAXIMIZED
                    ' L'azione è avvenuta possiamo notificarla
                End Select
                retVal = CallWindowProc(lpfnOld, hWnd, uMsg, wParam, lParam)

            Case WM_SIZING
                Dim rct          As RECT
                CopyMemory rct, ByVal lParam, LenB(rct)
                ' Se volessi impedire il movimento potrei forzare quì la
                ' posizione passando dei parametri fissi alla variabile
                ' strutturata mWNDPOS
                WITH rct
                    .Left = myLeft
                    .Top = myTop
                    .Right = myRight
                    .Bottom = myBottom
                END WITH
                CopyMemory ByVal lParam, rct, LenB(rct)
                retVal = CallWindowProc(lpfnOld, hWnd, uMsg, wParam, lParam)

            Case Else
                retVal = CallWindowProc(lpfnOld, hWnd, uMsg, wParam, lParam)
        End Select
        NewWindowProc = retVal

End Function
```

Riferimenti

Nei DEMO inseriti in questo Tutorial possiamo trovare in dettaglio l'uso di questi Messaggi:

Access97

SCD_Expand2MDI_97.mdb

AccessXP

SCD_Expand2MDI.mdb
SCD_AbilitaMessaggi.mdb

Questi sono in forma decisamente meno leggibile per la tecnica più avanzata
Advanced_SubClass_Simple.mdb
Nelle Form:
Expand2MDI
AbilitaMessaggi

DEMO_Sperimentale.mdb
Nelle Form:
Expand2MDI
AbilitaMessaggi

ELENCO MESSAGGI DI NOTIFICA PIU' UTILI

Il sistema per intercettarli è sempre quello, ma consiglio di leggere sempre le modalità di utilizzo su MSDN che deve rimanere la guida per la programmazione a basso livello.

```
'// General messages
Public Const WM_NULL = &H0
Public Const WM_CREATE = &H1
Public Const WM_DESTROY = &H2
Public Const WM_SETTEXT = &HC
Public Const WM_GETTEXT = &HD
Public Const WM_GETTEXTLENGTH = &HE
Public Const WM_QUIT = &H12
Public Const WM_SETCURSOR = &H20
Public Const WM_SETFONT = &H30
Public Const WM_GETFONT = &H31
Public Const WM_GETOBJECT = &H3D
Public Const WM_COPYDATA = &H4A
Public Const WM_NOTIFY = &H4E
Public Const WM_HELP = &H53
Public Const WM_NOTIFYFORMAT = &H55
    Public Const NFR_ANSI = 1
    Public Const NFR_UNICODE = 2
    Public Const NF_QUERY = 3
    Public Const NF_REQUERY = 4
Public Const WM_GETICON = &H7F
Public Const WM_SETICON = &H80
    Public Const ICON_SMALL = 0
    Public Const ICON_BIG = 1
Public Const WM_SYNCPAINT = &H88
Public Const WM_COMMAND = &H111
Public Const WM_TIMER = &H113
Public Const WM_PARENTNOTIFY = &H210
Public Const WM_DROPFILES = &H233
Public Const WM_APP = &H8000

'// WM_SYSCOMMAND and Publics
Public Const WM_SYSCOMMAND = &H112
    '// Values for wParam in WM_SYSCOMMAND
'=============================================================================
'                              ATTENZIONE
'=============================================================================
'                 Quì ACCESS evidenzia delle anomalie
    'Per Intercettarlo occorre ridefinirlo.
    Public Const SC_SIZE = &HF000&
    'Non so come ma il SC_SIZE è attivo per
    'valori da &HF0001& ÷ &HF0008& .
'=============================================================================
    'Per Intercettarlo occorre ridefinirlo.
    'Public Const SC_MOVE = &HF010&
    Public Const SC_MOVE = &HF012&
'=============================================================================
    Public Const SC_MINIMIZE = &HF020&
    Public Const SC_MAXIMIZE = &HF030&
    Public Const SC_NEXTWINDOW = &HF040&
    Public Const SC_PREVWINDOW = &HF050&
    Public Const SC_CLOSE = &HF060&
    Public Const SC_VSCROLL = &HF070&
    Public Const SC_HSCROLL = &HF080&
    Public Const SC_MOUSEMENU = &HF090&
```

```
        Public Const SC_KEYMENU = &HF100&
        Public Const SC_ARRANGE = &HF110&
        Public Const SC_RESTORE = &HF120&
        Public Const SC_TASKLIST = &HF130&
        Public Const SC_SCREENSAVE = &HF140&
        Public Const SC_HOTKEY = &HF150&

    Public Const WM_USER = &H400

    '// Activation and focus
    Public Const WM_ACTIVATE = &H6
        '//WM_ACTIVATE state values
        Public Const WA_INACTIVE = 0
        Public Const WA_ACTIVE = 1
        Public Const WA_CLICKACTIVE = 2

    Public Const WM_SETFOCUS = &H7
    Public Const WM_KILLFOCUS = &H8
    Public Const WM_ENABLE = &HA
    Public Const WM_CLOSE = &H10
    Public Const WM_SHOWWINDOW = &H18
    Public Const WM_ACTIVATEAPP = &H1C
    Public Const WM_MOUSEACTIVATE = &H21
        '// WM_MOUSEACTIVEATE codes
        Public Const MA_ACTIVATE = 1
        Public Const MA_ACTIVATEANDEAT = 2
        Public Const MA_NOACTIVATE = 3
        Public Const MA_NOACTIVATEANDEAT = 4

    Public Const WM_CHILDACTIVATE = &H22
    Public Const WM_CAPTURECHANGED = &H215
    Public Const WM_SETHOTKEY = &H32
    Public Const WM_GETHOTKEY = &H33
    Public Const WM_HOTKEY = &H312
        '// Keyboard modifiers for WM_HOTKEY
        Public Const MOD_ALT = &H1
        Public Const MOD_CONTROL = &H2
        Public Const MOD_SHIFT = &H4
        Public Const MOD_WIN = &H8
        Public Const MOD_WIN = &H8

    '// Sizing and movement
    Public Const WM_MOVE = &H3
    Public Const WM_SIZE = &H5
        Public Const SIZE_RESTORED = 0
        Public Const SIZE_MINIMIZED = 1
        Public Const SIZE_MAXIMIZED = 2
        Public Const SIZE_MAXSHOW = 3
        Public Const SIZE_MAXHIDE = 4

    Public Const WM_QUERYOPEN = &H13
    Public Const WM_GETMINMAXINFO = &H24
    Public Const WM_ENTERSIZEMOVE = &H231
    Public Const WM_EXITSIZEMOVE = &H232
    Public Const WM_WINDOWPOSCHANGING = &H46
    Public Const WM_WINDOWPOSCHANGED = &H47

    Public Const WM_SIZING = &H214
        Public Const WMSZ_LEFT = 1
        Public Const WMSZ_RIGHT = 2
        Public Const WMSZ_TOP = 3
```

```
        Public Const WMSZ_TOPLEFT = 4
        Public Const WMSZ_TOPRIGHT = 5
        Public Const WMSZ_BOTTOM = 6
        Public Const WMSZ_BOTTOMLEFT = 7
        Public Const WMSZ_BOTTOMRIGHT = 8
    Public Const WM_MOVING = &H216

    '// Drawing and painting
    Public Const WM_SETREDRAW = &HB
    Public Const WM_PAINT = &HF
    Public Const WM_COMPAREITEM = &H39
    Public Const WM_QUERYDRAGICON = &H37
    Public Const WM_ERASEBKGND = &H14
    Public Const WM_PAINTICON = &H26
    Public Const WM_ICONERASEBKGND = &H27
    Public Const WM_DRAWITEM = &H2B
    Public Const WM_MEASUREITEM = &H2C
    Public Const WM_DELETEITEM = &H2D
    Public Const WM_VKEYTOITEM = &H2E
    Public Const WM_CHARTOITEM = &H2F

    '// Environment messages
    Public Const WM_COMPACTING = &H41
    Public Const WM_COMMNOTIFY = &H44
    Public Const WM_FONTCHANGE = &H1D
    Public Const WM_TIMECHANGE = &H1E
    Public Const WM_QUERYENDSESSION = &H11
    Public Const WM_SYSCOLORCHANGE = &H15
    Public Const WM_ENDSESSION = &H16
    Public Const WM_WININICHANGE = &H1A
    Public Const WM_SETTINGCHANGE = WM_WININICHANGE
    Public Const WM_DEVMODECHANGE = &H1B
    Public Const WM_DEVICECHANGE = &H219
    Public Const WM_INPUTLANGCHANGEREQUEST = &H50
    Public Const WM_INPUTLANGCHANGE = &H51
    Public Const WM_USERCHANGED = &H54
    Public Const WM_STYLECHANGING = &H7C
    Public Const WM_STYLECHANGED = &H7D
    Public Const WM_DISPLAYCHANGE = &H7E
    Public Const WM_ENTERIDLE = &H121

    Public Const WM_NCHITTEST = &H84
        '// NC_HITTEST area codes
        Public Const HTERROR = (-2)
        Public Const HTTRANSPARENT = (-1)
        Public Const HTNOWHERE = 0
        Public Const HTCLIENT = 1
        Public Const HTCAPTION = 2
        Public Const HTSYSMENU = 3
        Public Const HTGROWBOX = 4
        Public Const HTSIZE = HTGROWBOX
        Public Const HTMENU = 5
        Public Const HTHSCROLL = 6
        Public Const HTVSCROLL = 7
        Public Const HTMINBUTTON = 8
        Public Const HTMAXBUTTON = 9
        Public Const HTLEFT = 10
        Public Const HTRIGHT = 11
        Public Const HTTOP = 12
        Public Const HTTOPLEFT = 13
        Public Const HTTOPRIGHT = 14
```

```
        Public Const HTBOTTOM = 15
        Public Const HTBOTTOMLEFT = 16
        Public Const HTBOTTOMRIGHT = 17
        Public Const HTBORDER = 18
        Public Const HTREDUCE = HTMINBUTTON
        Public Const HTZOOM = HTMAXBUTTON
        Public Const HTSIZEFIRST = HTLEFT
        Public Const HTSIZELAST = HTBOTTOMRIGHT
        Public Const HTOBJECT = 19
        Public Const HTCLOSE = 20
        Public Const HTHELP = 21

    Public Const WM_NCPAINT = &H85
    Public Const WM_NCACTIVATE = &H86
    Public Const WM_NCMOUSEMOVE = &HA0
    Public Const WM_NCLBUTTONDOWN = &HA1
    Public Const WM_NCLBUTTONUP = &HA2
    Public Const WM_NCLBUTTONDBLCLK = &HA3
    Public Const WM_NCRBUTTONDOWN = &HA4
    Public Const WM_NCRBUTTONUP = &HA5
    Public Const WM_NCRBUTTONDBLCLK = &HA6
    Public Const WM_NCMBUTTONDOWN = &HA7
    Public Const WM_NCMBUTTONUP = &HA8
    Public Const WM_NCMBUTTONDBLCLK = &HA9

    '// Keyboard messages
    Public Const WM_KEYFIRST = &H100
    Public Const WM_KEYDOWN = &H100
    Public Const WM_KEYUP = &H101
    Public Const WM_CHAR = &H102
    Public Const WM_DEADCHAR = &H103
    Public Const WM_SYSKEYDOWN = &H104
    Public Const WM_SYSKEYUP = &H105
    Public Const WM_SYSCHAR = &H106
    Public Const WM_SYSDEADCHAR = &H107
    Public Const WM_KEYLAST = &H108

    '// Scrolling
    Public Const WM_HSCROLL = &H114
    Public Const WM_VSCROLL = &H115

    '// Menu messages
    Public Const WM_INITMENU = &H116
    Public Const WM_INITMENUPOPUP = &H117
    Public Const WM_MENUSELECT = &H11F
    Public Const WM_MENUCHAR = &H120
    Public Const WM_MENURBUTTONUP = &H122
    Public Const WM_MENUDRAG = &H123
    Public Const WM_MENUGETOBJECT = &H124
    Public Const WM_UNINITMENUPOPUP = &H125
    Public Const WM_MENUCOMMAND = &H126
    Public Const WM_ENTERMENULOOP = &H211
    Public Const WM_EXITMENULOOP = &H212
    Public Const WM_NEXTMENU = &H213
    Public Const WM_CONTEXTMENU = &H7B

    '// MDI-related messages
    Public Const WM_MDICREATE = &H220
    Public Const WM_MDIDESTROY = &H221
    Public Const WM_MDIACTIVATE = &H222
    Public Const WM_MDIRESTORE = &H223
```

```
Public Const WM_MDINEXT = &H224
Public Const WM_MDIMAXIMIZE = &H225
Public Const WM_MDITILE = &H226
Public Const WM_MDICASCADE = &H227
Public Const WM_MDIICONARRANGE = &H228
Public Const WM_MDIGETACTIVE = &H229
Public Const WM_MDISETMENU = &H230
Public Const WM_MDIREFRESHMENU = &H234

'// Mouse messages
Public Const WM_MOUSEFIRST = &H200
Public Const WM_MOUSEMOVE = &H200
Public Const WM_LBUTTONDOWN = &H201
Public Const WM_LBUTTONUP = &H202
Public Const WM_LBUTTONDBLCLK = &H203
Public Const WM_RBUTTONDOWN = &H204
Public Const WM_RBUTTONUP = &H205
Public Const WM_RBUTTONDBLCLK = &H206
Public Const WM_MBUTTONDOWN = &H207
Public Const WM_MBUTTONUP = &H208
Public Const WM_MBUTTONDBLCLK = &H209
Public Const WM_MOUSEWHEEL = &H20A
Public Const WM_MOUSELAST = &H20A
Public Const WM_MOUSEHOVER = &H2A1    '// WINVER >= 2K or COMMCTL >=4.71
Public Const WM_MOUSELEAVE = &H2A3    '// WINVER >= 2K or COMMCTL >=4.71
    '//Key State Masks for Mouse Messages
    Public Const MK_LBUTTON = &H1
    Public Const MK_RBUTTON = &H2
    Public Const MK_SHIFT = &H4
    Public Const MK_CONTROL = &H8
    Public Const MK_MBUTTON = &H10

'// Edit messages (as in general editing, not the control)
Public Const WM_CUT = &H300
Public Const WM_COPY = &H301
Public Const WM_PASTE = &H302
Public Const WM_CLEAR = &H303
Public Const WM_UNDO = &H304
'// Clipboard messages
Public Const WM_RENDERFORMAT = &H305
Public Const WM_RENDERALLFORMATS = &H306
Public Const WM_DESTROYCLIPBOARD = &H307
Public Const WM_DRAWCLIPBOARD = &H308
Public Const WM_PAINTCLIPBOARD = &H309
Public Const WM_VSCROLLCLIPBOARD = &H30A
Public Const WM_SIZECLIPBOARD = &H30B
Public Const WM_ASKCBFORMATNAME = &H30C
Public Const WM_CHANGECBCHAIN = &H30D
Public Const WM_HSCROLLCLIPBOARD = &H30E
```

HANDLE DEI CONTROLLI IN ACCESS

I controlli di Access non sono come quelli standard di VB, sono disegnati sullo schermo a RUNTIME, e come tali, diversamente dai comandi di VB, non hanno un hWnd univoco.
Quando un controllo di Access su una Maschera riceve il fuoco, si trasforma in una finestra ed è possibile richiamarne l'Handle usando il GetFocus api.

Questa funzione scritta originariamente da *Dev Ashish* restituisce l'Handle del controllo spostando il Focus preventivamente:

http://www.mvps.org/access/api/api0027.htm

CODICE

```
Private Declare Function GetFocus Lib "user32" _
                         Alias "GetFocus" () As Long

Function fhWnd(ctl As Control) As Long
    On Error Resume Next
    ctl.SetFocus
    If Err Then
        fhWnd = 0
    Else
        fhWnd = GetFocus
    End If
    On Error GoTo 0
End Function
```

LEGENDA

Finestra
Per finestra non si intende solo la "form", bensì qualsiasi oggetto che compone l'interfaccia: un pulsante è una finestra, una casella combinata è una finestra, il Desktop è una finestra... Tutte le finestre appartengono ad una Classe e ciascuna classe ha delle caratteristiche proprie ed altre comuni a più classi. Quando viene creato un pulsante, ad esempio, viene semplicemente creata un'istanza (copia) della classe Button (ed è quello che viene definito Oggetto)

Handle
L'handle di un oggetto non è altro che un numero intero a 32 bit che identifica univocamente l'oggetto. Poiché ci possono essere altre istanze della stessa classe; poiché ci potrebbe essere un'istanza della classe Oggetto con lo stesso nome l'handle è come se fosse il "Codice fiscale" del nostro oggetto, ovvero un codice che il sistema operativo gli assegna all'atto della sua creazione(istanza).

Evento
E' una procedura o funzione scatenata da un'azione dell'utente o del sistema operativo, o a fronte di una richiesta esterna.

API
API, *Application Programming Interface*, non sono altro che le funzioni di Windows, ovvero una serie di funzioni messe a disposizione dal Sistema Operativo (quindi contenute nelle DLL con esso distribuite) alle quali possono far riferimento tutte le applicazioni.

AddressOf
L'operatore AddressOf permette di specificare che si sta passando come argomento una funzione definita dall'utente (detta funzione **callback**). AddressOf necessita del rispetto di alcune condizioni:

1) *Può essere usato solo con funzioni (o procedure) definite dall'utente.*
2) *Le funzioni devono trovarsi nello stesso progetto.*
3) *Le funzioni devono essere dichiarate in un modulo standard.*

ByVal
E' una parola chiave che consente di passare il valore di un argomento anziché il suo indirizzo di memoria. Ciò consente alla Routine di accedere ad una copia dell'Argomento di conseguenza il valore non può essere modificato.

ByRef
E' una parola chiave che consente di passare l'indirizzo di un argomento anziché il suo valore. Ciò consente alla Routine di accedere al valore stesso puntando alla locazione di residenza e di poterlo modificare.

Declare
Utilizzata a livello di modulo per dichiarare riferimenti a routine esterne in una libreria a collegamento dinamico (DLL).