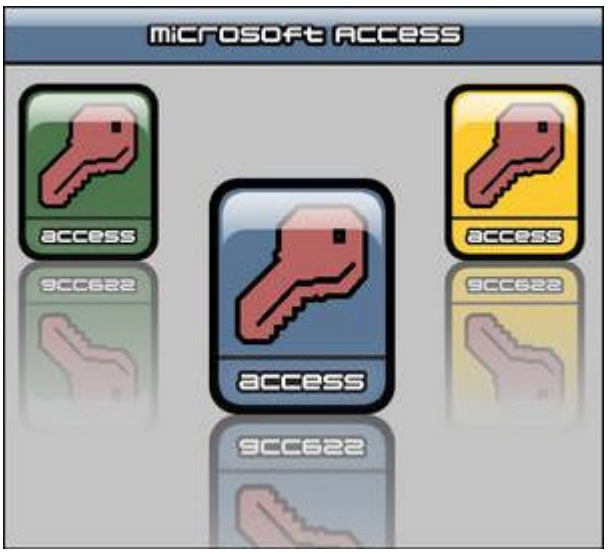
API TUTORIAL



MS ACCESS

INDICE

TUTORIAL API

API è l'acronimo di ***Application Programming Interface*** (interfaccia per la programmazione di applicativi). Si tratta (senza alcuna pretesa di essere esaustivi) di una serie di funzioni del sistema operativo che possono essere utilizzate da tutti i programmi che ne facciano richiesta mediante opportune istruzioni. Queste funzioni sono, di solito, contenute in alcuni file ***DLL*** (***Dinamic Link Library***, libreria a collegamento dinamico) di Windows, come ***KERNEL32.DLL***, ***USER32.DLL*** o ***GDI32.DLL***, e sono molto utili: a volte, ad esempio, diventano indispensabili per implementare alcune caratteristiche nel software.
Le API vengono utilizzate nella maggior parte dei linguaggi che vi consentono di fare chiamate a livello di sistema. La nota positiva delle API è che una volta che avete imparato a fare chiamate in un linguaggio, passare ad un altro linguaggio per eseguire le stesse chiamate, non richiede grosse modifiche.
In particolare questo discorso si presta tra VB e VBA.

Per poter richiamare da VBA una funzione disponibile in una DLL, è necessario fornire a VBA informazioni sulla posizione in cui si trova tale funzione e su come richiamarla. A tale scopo è possibile procedere in due modi, ovvero impostare un riferimento alla libreria dei tipi della DLL oppure utilizzare un'istruzione Declare in un modulo.
La prima modalità, ovvero l'impostazione di un riferimento alla libreria dei tipi di una DLL, è la più semplice per l'utilizzo delle funzioni disponibili nella DLL. Quando si imposta il riferimento, è possibile richiamare la funzione di DLL come se facesse parte del progetto che si sta sviluppando. È però necessario tenere presenti un paio di raccomandazioni. In primo luogo, l'impostazione di riferimenti a più librerie dei tipi può influire sulle prestazioni dell'applicazione. In secondo luogo, non tutte le DLL includono librerie dei tipi. Sebbene sia possibile impostare un riferimento a una DLL senza una libreria dei tipi, non è possibile richiamare le funzioni di tale DLL come se facessero parte del progetto che si sta sviluppando.

VBA è un potente strumento per la creazione di applicazioni di Windows, tuttavia, consente di controllare solo una parte circoscritta del sistema operativo. Si tratta della parte resa disponibile dalle funzioni e dagli oggetti esposti direttamente a VBA nell'applicazione che si sta sviluppando.
Ad esempio, in VBA sono disponibili funzioni in grado di leggere e scrivere in una parte del Registro di sistema riservata alle applicazioni VBA. Queste funzioni, ovvero *GetSetting*, *GetAllSettings*, *SaveSetting* e *DeleteSetting*, consentono di gestire le informazioni sull'applicazione che si sta sviluppando tra le varie sessioni. Mediante queste funzioni, tuttavia, è possibile agire su una sola sottochiave del Registro di sistema e precisamente *\HKEY_CURRENT_USER\Software\VB* e *VBA* Program Settings. Per archiviare o recuperare informazioni da altre sezioni del Registro di sistema, è necessario utilizzare l'API di Windows.
Per modificare il Registro di sistema sono disponibili varie funzioni API che è possibile utilizzare congiuntamente.
Ai programmatori VBA possono inoltre risultare utili le funzioni API che consentono di modificare gli Appunti di Windows. In VBA non è disponibile un oggetto per l'utilizzo degli Appunti, ma è possibile eseguire il wrapping delle funzioni API per gli Appunti in un modulo di classe per creare un oggetto personalizzato semplice e riutilizzabile che rappresenti gli Appunti di Windows.
Per eseguire una singola operazione è consigliabile utilizzare contemporaneamente più funzioni. Ad esempio, la funzione *OpenClipboard* apre gli Appunti per consentire di esaminarne il contenuto e ne impedisce la modifica da parte di altre applicazioni. La funzione *GetClipboardData* restituisce i dati salvati negli Appunti, mentre la funzione *CloseClipboard* chiude gli Appunti rendendoli nuovamente disponibili per altre applicazioni.

Link Completo:
http://msdn.microsoft.com/library/ita/default.asp?url=/library/ITA/modcore/html/deovrretrievingerrorinformationfollowingdllfunctioncalls.asp

Link abbreviato:
http://tinyurl.com/7upoy

HANDLE

Un altro importante concetto con cui è necessario familiarizzare prima di richiamare funzioni di DLL è quello di handle. Un handle è un valore integer positivo a 32 bit utilizzato da Microsoft® Windows® per identificare una finestra o un altro oggetto, ad esempio un tipo di carattere o una bitmap.
In Windows una finestra può essere rappresentata da oggetti di vario tipo. Una finestra può essere un'area rettangolare delimitata dello schermo, come ad esempio le note finestre di applicazione, oppure un controllo di un form, ad esempio una casella di riepilogo o una barra di scorrimento, sebbene non tutti i tipi di controlli possano essere finestre. Anche le icone visualizzate sul desktop e il desktop stesso sono finestre.
Poiché tutti questi tipi di oggetti sono finestre, possono essere trattati da Windows in modo analogo. Windows assegna a ogni finestra un handle univoco, che utilizza per gestire tale finestra. Molte funzioni API restituiscono handle o accettano handle come argomenti.
L'handle viene assegnato quando la finestra viene creata e rilasciato quando la finestra viene eliminata. Anche se l'handle rimane invariato per tutto il ciclo di vita della finestra, in caso di eliminazione e ricreazione di una finestra non esistono garanzie che l'handle rimanga lo stesso. Pertanto, se si archivia un handle in una variabile, l'handle non sarà più valido in caso di eliminazione della finestra.

LE DLL

Come dicevamo sono delle librerie *Dinamic Link Library*, a collegamento dinamico, ovvero collegate Runtime.
Il vantaggio di queste librerie è che non vengono caricate interamente, ma viene caricata singolarmente la Routine dichiarata, con un notevole risparmio di memoria.
Le DLL più usate sono:

```
KERNEL32.DLL      '// File manipulation functions
USER32.DLL        '// Window attribute and creation functions
GDI32.DLL         '// Graphical and Drawing functions
SHELL32.DLL       '// Shell library functions
```

Quindi le DLL non sono altro che dei contenitori di Funzioni specializzate divise per settore che interagiscono a livello di Sistema Operativo.
L'interprete di linguaggio(VBA nel nostro caso) quando espone delle funzioni non fà altro che presentarci un metodo semplificato per accedere alla primitiva API.
Spesso però questo sistema introduce forti rallentamenti così che alcune volte è preferibile escludere l'interprete.
La cosa più comune però è quella di avere accesso a tutta una serie di informazioni e di impostazioni del SO che non sarebbero accessibili con il nostro linguaggio di programmazione.

http://msdn.microsoft.com/library/ita/default.asp?url=/library/ITA/modcore/html/deovrretrievingerrorinformationfollowingdllfunctioncalls.asp

Nonostante vi siano molte analogie tra le chiamate alle funzioni di DLL e le chiamate alle funzioni VBA (Microsoft® Visual Basic®, Applications Edition), è

opportuno chiarire alcune differenze che potrebbero rendere complicato l'apprendimento dei meccanismi utilizzati per richiamare le funzioni di DLL. In questa sezione vengono illustrati i tipi di dati e i prefissi utilizzati dagli argomenti nelle funzioni di DLL. Viene inoltre illustrato come restituire una stringa, come passare una struttura di dati.

ISTRUZIONE DECLARE

Questa istruzione è la parte fondamentale per la dichiarazione delle Sub o Function API, è utilizzata a livello di modulo per dichiarare riferimenti a routine esterne in una libreria a collegamento dinamico (DLL).
Comprendere la modalità di utilizzo di questa istruzione è importante per discriminare il modo di utilizzo delle chiamate API

Sub

[**Public** | **Private**] **Declare Sub** *nome* **Lib** "*nomelibreria*" [**Alias** "*nomealias*"] [**(**[*elencoargomenti*]**)**]

Function

[**Public** | **Private**] **Declare Function** *nome* **Lib** "*nomelibreria*" [**Alias** "*nomealias*"] [**(**[*elencoargomenti*]**)**] [**As** *tipo*]

SINTASSI DLL

La sintassi dell'istruzione **Declare** è composta dalle seguenti parti:

Parte	Descrizione
Public	Facoltativa. Dichiara le routine disponibili per tutte le altre routine in tutti i moduli.
Private	Facoltativa. Dichiara le routine disponibili solo nei moduli in cui viene fatta la dichiarazione.
Sub	Facoltativa (è comunque necessario specificare **Sub** o **Function**). Indica che la routine non restituisce alcun valore.
Function	Facoltativa (è comunque necessario specificare **Sub** o **Function**). Indica che la routine restituisce un valore che può essere utilizzato in una espressione.
nome	Obbligatoria. Qualsiasi nome di routine valido. Si noti che i punti di ingresso delle DLL distinguono tra maiuscole e minuscole.
Lib	Obbligatoria. Indica che una DLL o una risorsa codice contiene la routine da dichiarare. La proposizione **Lib** è obbligatoria in tutte le dichiarazioni.
nomelibreria	Obbligatoria. Nome della DLL o della risorsa codice che contiene la routine dichiarata.
Alias "Pseudonimo"	Facoltativa. Indica che la routine da richiamare ha un nome diverso nella DLL. Questo risulta utile se il nome della routine esterna corrisponde a una parola chiave. È inoltre possibile utilizzare l'argomento **Alias** se una routine in una DLL ha lo stesso nome di una variabile pubblica, di una costante o di qualsiasi altra routine nella stessa area di validità.
nomealias	Facoltativa. Nome della routine nella DLL o nella risorsa codice. Se il primo carattere non è #, *nomealias* è il nome del punto di ingresso nella routine nella DLL. Se # è il primo carattere, tutti i caratteri che seguono devono indicare il numero ordinale del punto di ingresso nella routine.
elencoargomenti	Facoltativa. Elenco di variabili che rappresentano argomenti passati alla routine alla chiamata: [**Optional**] [**ByVal** | **ByRef**] [**ParamArray**] *nomevariabile*[**( )**] [**As** *tipo*]
tipo	Facoltativa. Tipo di dati del valore restituito dalla routine **Function**. L'argomento può essere Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal (non ancora supportato), Date, String (solo di lunghezza variabile), Variant oppure un tipo definito dall'utente o un tipo di oggetto.

SINTASSI API

Per essere utilizzate, è necessario che l'ambiente di sviluppo, Visual Basic o C++ che sia, sappia come trattarle; questa operazione è chiamata *dichiarazione*. Le dichiarazioni di funzioni sono, a volte, molto lunghe e difficili da comprendere. Anche solo un piccolo errore, oltre ad impedire la corretta esecuzione della funzione stessa, può costringere a terminare l'applicazione o, nel peggiore dei casi, a riavviare Windows.

PRIVATE o PUBLIC

Supponiamo di dichiarare una funzione in un modulo della Maschera, se la dichiarazione non è definita Private VBA vi restituirà un errore di compilazione e vi dirà che le dichiarazione non sono permesse come membri pubblici in moduli di Classe(la Form è una Classe con interfaccia grafica).
Non dimenticate che la funzione sarà visibile soltanto all'interno del modulo di classe. Una dichiarazione Public può essere effettuata solo ed esclusivamente in un modulo standard.
In alcuni casi, potete ottenere un messaggio di Nome Ambiguo o Non Univoco rilevato da VBA, questo significa che avete due funzioni, costanti o qualunque altra cosa dichiarata più di una volta nel progetto e visibile a tutti i livelli , pertanto si tratta di dichiarazione Public.

L'ALIAS

La maggior parte delle funzioni hanno uno "pseudonimo" detto alias, il che significa che possono avere nomi differenti dall'originale semplicemente usando la Clausola Alias.(vedi più avanti l'esempio di utilizzo dell'Alias)

Un esempio di quanto dicevamo è questo nel quale facciamo riferimento sempre alla funzione SendMessageA ma usiamo un Alias differente per diversificare l'uso della chiamata(come vedremo in seguito nell'esempio).

```
Private Declare Function SendMessage Lib "user32" _
                         Alias "SendMessageA" _
                         (ByVal hwnd           As Long, _
                         ByVal wMsg            As Long, _
                         ByVal wParam          As Long, _
                         ByVal lParam          As String) As Long

Private Declare Function SendMessageSTRING Lib "user32" _
                         Alias "SendMessageA" _
                         (ByVal hwnd           As Long, _
                         ByVal wMsg            As Long, _
                         ByVal wParam          As Long, _
                         ByVal lParam          As String) As Long

Private Declare Function SendMessageLONG Lib "user32" _
                         Alias "SendMessageA" _
                         (ByVal hwnd           As Long, _
                         ByVal wMsg            As Long, _
                         ByVal wParam          As Long, _
                         ByVal lParam          As Long) As Long
```

VARIABILI STRUTTURATE

Per molte funzioni di DLL è necessario passare una struttura di dati utilizzando un formato predefinito. Quando si richiama una funzione di DLL da VBA, si passa un tipo di dati definito dall'utente scelto in base ai requisiti della funzione. Per determinare quando è necessario passare un tipo di dati definito dall'utente e la definizione del tipo da includere nel codice, è possibile esaminare l'istruzione Declare della funzione. Un argomento per cui è necessaria una struttura di dati viene sempre dichiarato come puntatore di tipo long, ovvero come un valore numerico a 32 bit che punta alla struttura di dati in memoria. Il prefisso convenzionale per un argomento dichiarato come puntatore di tipo long è "lp". Inoltre, il tipo di dati dell'argomento corrisponde al nome della struttura di dati.

ESEMPIO

```
Public Declare Function GetWindowRect Lib "user32" _
                        (ByVal hWnd     As Long, _
                         lpRect         As RECT) As Long
```

La variabile di tipo **RECT** in questo caso è una variabile strutturata, è come avere un contenitore nel quale inseriamo diversi tipi di variabili, quindi attribuiamo un nome al contenitore.

In questo caso il contenitore è **RECT** ed al suo interno abbiamo 4 variabili di tipo Long che chiameremo *Atomi* o *Membri* della struttura. La variabile RECT rappresenta le coordinate rettangolari relative ad un oggetto.

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

PASSAGGIO STRINGHE

Le funzioni di DLL non restituiscono le stringhe allo stesso modo delle funzioni VBA (Microsoft® Visual Basic®, Applications Edition). Poiché le stringhe vengono sempre passate alle funzioni di DLL per riferimento, la funzione di DLL può modificare il valore dell'argomento stringa. Anziché restituire una stringa come valore restituito della funzione, come accadrebbe probabilmente in VBA, una funzione di DLL restituisce una stringa in un argomento di tipo String passato alla funzione. Il valore restituito effettivo della funzione è spesso un valore integer di tipo long che specifica il numero di byte scritti nell'argomento stringa.
Una funzione di DLL che accetta un argomento stringa riceve un puntatore alla posizione di tale stringa nella memoria. Un puntatore è un indirizzo di memoria che indica dove la stringa è archiviata. Pertanto, quando si passa una stringa a una funzione di DLL da VBA, in realtà si passa alla funzione di DLL un puntatore per la stringa in memoria. La funzione di DLL modifica quindi la stringa archiviata in corrispondenza di tale indirizzo.
Per richiamare una funzione di DLL che scrive in una variabile String, è necessario eseguire una procedura aggiuntiva per formattare la stringa in modo corretto. Innanzitutto, è necessario che la variabile String sia una stringa che termina con un valore Null. Una stringa di questo tipo termina con un carattere Null speciale, specificato dalla costante *vbNullChar* VBA.
In secondo luogo, una funzione di DLL non può modificare la dimensione di una stringa dopo che è stata creata. Pertanto, è necessario assicurarsi che la stringa passata a una funzione sia sufficientemente grande da poter contenere l'intero valore restituito. Quando si passa una stringa a una funzione di DLL, in genere è necessario specificare la dimensione della stringa passata in un altro argomento. In Microsoft® Windows® viene tenuta traccia della lunghezza della stringa per garantire che non venga sovrascritta memoria utilizzata dalla stringa.
Un valido modo per passare una stringa a una funzione di DLL consiste nel creare una variabile String e nell'utilizzare la funzione String$ per inserire al suo interno caratteri Null, in modo da creare spazio sufficiente per la stringa restituita dalla funzione. Ad esempio, in questo frammento di codice viene creata una stringa di 144 byte contenente caratteri Null:

ESEMPIO

```
    Dim strTempPath    As String

    strTempPath = String$(144, vbNullChar)
```

MESSAGGI

Ora conoscete che cosa sono le funzioni api, ma per capire come funzionano e/o come si usano occorre entrare nel cuore della struttura e capire cosa sono i Messaggi.
I messaggi sono il metodo di base con il quale Windows dice al vostro programma che un certo genere evento si è verificato ed è necessario processarlo. Un messaggio alla vostra form è trasmesso quando l'utente clicca un CommandButton o si sposta su esso o semplicemente quando seleziona una TextBox e scrive del testo. Tutti i messaggi sono trasmessi con quattro parametri:

Un Handle della finestra; Hwnd

Un contrassegno di messaggio; Msg

Due valori (lunghi) 32-bit. lParam/wParam

L'Handle della finestra è esattamente il riferimento alla stessa alla quale è inviato il messaggio. Il contrassegno è realmente il tipo di evento che si è verificato (Click, Mousemove ecc...) i due valori specificano le informazioni supplementari per il messaggio (come dove è il cursore del mouse quando il mouse viene spostato).
Così, quando l'utente sposta il mouse sopra la vostra form, Windows trasmette WM_MOUSEMOVE alla vostra finestra e VB ottiene il messaggio ed i relativi parametri in modo tale da poter eseguire il codice che avete impostato per l'evento di Form_MouseMove.

Una cosa importante da dire è che i messaggi possono essere trasmessi dal proprio applicativo verso le proprie o altre finestre individuate dall'Hwnd. La chiamata da usare è SendMessage o PostMessage (SendMessage indurrà la finestra a processare immediatamente il messaggio accodandolo alla coda messaggi ed attendendone l'esecuzione, mentre PostMessage è una funzione asincrona che accoda il messaggio poi ritorna senza attendere l'esito). Dovete specificare l'Handle della finestra alla quale trasmettere il messaggio tutti i messaggi sono disponibili come costanti nel visore del testo di VB api così come i valori corrispondenti.

FUNZIONI API

FINDEXECUTABLE

Restituisce il nome e l'handle del file eseguibile(.exe) associato al nome del file specificato.

Sintassi

```
HINSTANCE FindExecutable(
    LPCTSTR        lpFile,
    LPCTSTR        lpDirectory,
    LPTSTR         lpResult
);
```

Parametri

lpFile
Indirizzo della stringa(a terminazione nulla) che specifica il nome del File.

lpDirectory
Indirizzo della stringa(a terminazione nulla) che specifica la Directory di default.

lpResult
Indirizzo del buffer per ricevere il nome file restituito dalla chiamata. Questo nome file è una stringa a terminazione nulla che specifica il file Eseguibile avviato come associazione al file specificato nel parametro *lpFile*. Questo parametro deve contenere un valore valido e non nullo, e sarà usato per contenere la lunghezza massima di MAX_PATH.

Valore di ritorno
Il valore restituito è > 32 se la funzione ha recuperato correttamente l'eseguibile, viceversa il valore sarà <=32 se si è verificato un errore.
Gli errori possono essere codificati come indicato nell'esempio sotto.

Link Completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/shellcc/platform/shell/reference/functions/findexecutable.asp

Link abbreviato:
http://tinyurl.com/d99xo

ESEMPIO

```
Const ERROR_NOASSOC = 31            ‘ Nessun file associato
Const ERROR_FILE_NOT_FOUND = 2&     ‘ File non trovato
Const ERROR_PATH_NOT_FOUND = 3&     ‘ Path non trovata
Const ERROR_BAD_FORMAT = 11&        ‘ Formato non coerente
Const ERROR_OUT_OF_MEM = 0          ‘ Out of Memory

Private Declare Function FindExecutable Lib "shell32.dll" _
                         Alias "FindExecutableA" _
                         (ByVal lpFile        As String, _
                          ByVal lpDirectory   As String, _
                          ByVal lpResult      As String) As Long

Const cMAX_PATH = 260

Function fFindEXE(stFile     As String, _
                  stDir      As String) As String
' Esempio d’uso:
'   ?fFindEXE("c:\test.xls","c:\")
'
Dim lpResult      As String
Dim lngret        As Long
' Preparo il Buffer della stringa adatto a contenere il Testo
lpResult = Space(cMAX_PATH)
lngret = FindExecutable(stFile, stDir, lpResult)

If lngret > 32 Then
     ' Se il risultato è >32 significa che ha trovato l’associazione
     ' al file ESEGUIBILE
     ' Essendo una stringa con terminatore Nullo lo rimuovo
     fFindEXE = Left$(lpResult, Instr(1,lpResult,Chr(0))-1)
Else
     Select Case lngret:
          Case ERROR_NOASSOC:             fFindEXE = "Error: No Association"
          Case ERROR_FILE_NOT_FOUND:      fFindEXE = "Error: File Not Found"
          Case ERROR_PATH_NOT_FOUND:      fFindEXE = "Error: Path Not Found"
          Case ERROR_BAD_FORMAT:          fFindEXE = "Error: Bad File Format"
          Case ERROR_OUT_OF_MEM:          fFindEXE = "Error: Out of Memory"
     End Select
End If
End Function
```

GETCOMPUTERNAME

Restituisce il nome del Computer Locale recuperato dal NetBIOS. Questo nome è stabilito allo StartUp del sistema e viene letto dal Registry.

Sintassi

```
BOOL GetComputerName(
    LPTSTR        lpBuffer,
    LPDWORD       lpnSize
);
```

Parametri

lpBuffer
: [out] Puntatore al buffer che riceve la stringa a terminazione nulla contenente il Nome del Computer oppure il nome del Server virtuale. La dimensione del buffer deve essere sufficientemente larga da contenere MAX_COMPUTERNAME_LENGTH + 1 carattere.

lpnSize
: [in, out] In ingresso specifica la dimensione del Buffer in Caratteri. In uscita, il numero di caratteri copiati nel Buffer di destinazione ad esclusione del carattere Nullo, in pratica la lunghezza del Nome.
Se il Buffer è troppo piccolo, la funzione fallisce e ritorna un errore ERROR_BUFFER_OVERFLOW

Valore Restituitos
Se la funzione ha successo restituisce un valore <>0.
Se fallisce restituisce 0.

Link Completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sysinfo/base/getcomputername.asp

Link abbreviato:
http://tinyurl.com/brsba

ESEMPIO

```
Public Declare Function GetComputerName Lib "kernel32" _
                        Alias "GetComputerNameA" _
                        (ByVal lpBuffer       As String, _
                        lpnSize               As Long) As Long

Function MachineName() As String
    'Returns the computername
    Dim lngLen         As Long
    Dim lngResultAs Long
    Dim strCompName    As String

    MachineName = vbNullString
    lngLen = 255
    ' Preparo il Buffer della stringa adatto a contenere il Testo
    strCompName = String$(lngLen, 0)

    lngResult = GetComputerName(strCompName, lngLen)
    If lngResult <> 0 Then
        ' Dato che il parametro lpnSize in uscita contiene il numero
        ' di caratteri della stringa possiamo usare questo per delimitarla.
        MachineName = Left$(strCompName, lngLen)
    End If
End Function
```

GETUSERNAME

Restituisce il nome dell'Utente associate al thread corrente.

Sintassi

```
BOOL GetUserName(
    LPTSTR        lpBuffer,
    LPDWORD       nSize
);
```

Parametri

lpBuffer

[out]] Puntatore al buffer che riceve la stringa a terminazione nulla contenente il Nome di Login dell'utente. Se il Buffer non è sufficientemente largo la funzione fallisce. La dimensione del Buffer (**UNLEN** + 1 caratteri) conterrà il numero di caratteri massimo incluso il carattere di terminazione nullo.

nSize

[in, out] In ingresso specifica la dimensione del Buffer in Caratteri. In uscita, il numero di caratteri copiati nel Buffer di destinazione incluso il terminatore Nullo.
Se il Buffer è troppo piccolo, la funzione fallisce e ritorna un errore ERROR_INSUFFICIENT_BUFFER
Se il parametro è più grande di 32767, la funzione fallisce e restituisce un errore ERROR_INSUFFICIENT_BUFFER.

Valore Restituitos
Se la funzione ha successo restituisce un valore <>0.
Se fallisce restituisce 0.

Link completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sysinfo/base/getusername.asp

Link abbreviato:
http://tinyurl.com/5wvbu

ESEMPIO

```
Public Declare Function GetUserName Lib "advapi32.dll" _
                        Alias "GetUserNameA" _
                        (ByVal lpBuffer      As String, _
                        nSize                As Long) As Long

Function UserName() As String
    'Returns the UserName
    Dim lngLen         As Long
    Dim lngResultAs Long
    Dim strUserName    As String

    UserName = vbNullString
    lngLen = 255
    ' Preparo il Buffer della stringa adatto a contenere il Testo
    strCompName = String$(lngLen, 0)
    lngResult = GetUserName(strUserName, lngLen)

    If lngResult <> 0 Then UserName = strUserName
End Function
```

FINDWINDOW

Questa funzione restituisce l'Handle della prima finestra(Top-Level) avente come ClassName e WindowsName(Caption) quelli passati come parametri stringa.
Questa funzione non cerca Finestre figlio o Child. La ricerca eseguita da questa funzione non è CaseSensitive. Al fine di individuare una Finestra Child occorre vedere l'uso della funzione FindWindowEx(anche se in Access è decisamente meno semplice, un controllo TextBox oppure una ListBox sono finestre figlio o Child, ma al contrario di VB ricevono il puntatore di Handle solo quando hanno lo stato attivo "Focus").

Sintassi

```
HWND FindWindow(
    LPCTSTR        lpClassName,
    LPCTSTR        lpWindowName
);
```

Parametri

lpClassName
Puntatore ad una stringa con terminatore Nullo[chr(0)] che specifica il nome della Classe o dell'atomo relativo della struttura ***WNDCLASS***.
L'atomo è un componente della variabile strutturata. In questo caso l'atomo essendo un Pointer deve essere nella parte bassa della word *lpClassName*; la parte alta deve essere a zero.
Se *lpClassName* punta ad una stringa, questa rappresenta il nome della classe della finestra. Il nome della classe può essere qualsiasi nome registrato attraverso **RegisterClass** o **RegisterClassEx**, oppure qualsiasi nome delle classi di controlli predefinite.
Se *lpClassName* è NULL, la funzione cerca qualsiasi finestra la cui Caption corrisponde al parametro *lpWindowName*.

lpWindowName
Puntatore ad una stringa con terminatore Nullo[chr(0)] che specifica il nome del TITOLO della finestra(Caption). Se questo parametro è nullo tutti i nomi delle finestre corrispondono.

Valore Restituito
Se la funzione ha successo viene restituito l'Handle della finestra che corrispondeva al ClassName o al NomeFinestra. Se la funzione fallisce ritorna valore nullo.

Link completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/windows/windowreference/windowfunctions/findwindow.asp
Link abbreviato:
http://tinyurl.com/4l87r

```
Private Declare Function FindWindow Lib "user32" _
                         Alias "FindWindowA" _
                         (ByVal lpClassName    As String, _
                         ByVal lpWindowName    As String) As Long
```

Note
Se il parametro *lpWindowName* non è nullo, **FindWindow** chiama la funzione GetWindowText per ottenere il nome della finestra per confronto. Per una descrizione dei potenziali problemi che ne derivano vedere le note della funzione **GetWindowText**.

```
    "notepad":            strClassName = "NOTEPAD"
    "excel":              strClassName = "XLMain"
    "word":               strClassName = "OpusApp"
    '------------------   ALCUNI CLASSNAME DI ACCESS  -----------------
    "access":             strClassName = "OMain"
       "MDIClient":       strClassName = "MDIClient"
       "FinestraDB":      strClassName = "ODb"
       "Form":            strClassName = "OForm"
       "ListBox":         strClassName = "OGrid"
       "BarreMenù/Str.":  strClassName = "MsoCommandBar"
    '------------------   -------------------------  -----------------
```

Esempio di utilizzo a seconda di che parametro si passa:

```
lHwnd = FindWindow(strClassName, vbNullString)
lHwnd = FindWindow(vbNullString, strAppName)    ' Dove strAppName è la Captino
```

FINDWINDOWEX

La funzione di **FindWindowEx** restituisce l'Handle di una finestra il cui nome della classe ed il nome della finestra corrispondono alle stringhe specificate. La funzione cerca le finestre Child(figlie), comincianti con la seguente figlia specificata. Questa funzione non effettua una ricerca case-sensitive.

Sintassi

```
HWND FindWindowEx(
     HWND           hwndParent,
     HWND           hwndChildAfter,
     LPCTSTR        lpszClass,
     LPCTSTR        lpszWindow
);
```

Parametri

hwndParent

Handle della finestra Parent nella quale deve essere effettuata la ricerca.
Se *hwndParent* è NULL, la funzione di default usa l'Handle del Desktop come finestra Padre. La funzione cerca tra le Finestre definite Child del DeskTop.
In Win2000 ed XP il parametro *hwndParent* può assumere un valore alternativo HWND_MESSAGE, in questo caso la funzione effettuerà la ricerca tra tutte finestre definite message-only windows(Sono finestre che spediscono semplicemente i messaggi.)

hwndChildAfter

Parametro raramente usato.
Indica l'Handle della finestra Child dalla quale iniziare la ricerca della finestra successiva in base all'ordine Z(numerale progressivo). La child deve essere una finestra figlia del *hwndParent*, non semplicemente una discendente.
Se *hwndChildAfter* è NULL, la ricerca inizia con la prima Child windows nell'ordine Z.
Notre che se entrambi *hwndParent* e *hwndChildAfter* sono NULL, la funzione ricercherà tutte le top-level e message-only windows.

lpClassName

Puntatore ad una stringa con terminatore Nullo[chr(0)] che specifica il nome della Classe o dell'atomo. L'atomo deve essere nella parte bassa della word *lpClassName*; la parte alta deve essere a zero.
Se *lpClassName* punta ad una stringa, questa rappresenta il nome della classe della finestra. Il nome della classe può essere qualsiasi nome registrato attraverso **RegisterClass** o **RegisterClassEx**, oppure qualsiasi nome delle classi di controlli predefinite.

lpWindowName

Puntatore ad una stringa con terminatore Nullo[chr(0)] che specifica il nome del TITOLO della finestra(Caption). Se questo parametro è nullo tutti i nomi delle finestre corrispondono.

Valore Restituito
Se la funzione ha successo viene restituito l'Handle della finestra che corrispondeva al ClassName o al NomeFinestra.
Se la funzione fallisce ritorna valore nullo.
Per un'informazione estesa degli errori vedere la chiamata a GetLastError.

Link completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/windows/windowreference/windowfunctions/findwindowex.asp
Link abbreviato:
http://tinyurl.com/achzu

```
Private Declare Function FindWindowEx Lib "user32" _
                         Alias "FindWindowExA" _
                         (ByVal hWnd1          As Long, _
                         ByVal hWnd2           As Long, _
                         ByVal lpsz1           As String, _
                         ByVal lpsz2           As String) As Long
```

SENDMESSAGE/POSTMESSAGE

La funzione SendMessage invia messaggi specifici ad una finestra(hwnd). Chiama la procedura della Finestra ed attende che questa abbia processato il messaggio. La funzione PostMessage invece accoda il messaggio alla coda messaggi della finestra e rientra senza attenderne l'esecuzione.
La scelta tra le 2 funzioni quindi riguarda solo la possibilità di poter conoscere o meno l'esito del processamento piuttosto che solo l'esito della chiamata.

Sintassi

```
LRESULT SendMessage(
    HWND                hWnd,
    UINT                Msg,
    WPARAM              wParam,
    LPARAM              lParam
);
```

Parametri

hWnd
[in] Handle della Finestra che deve ricevere il messaggio da processare. Se questo parametro è HWND_BROADCAST, il messaggio viene inviato a tutte le Finestre di sistema in primo piano, incluse quelle disabilitate, non visibili, sovrapposte o popup ma non alle Finestre Child.

Msg
[in] Specifica il messaggio da inviare.

wParam
[in] Parametro addizionale specifico e dipendente dal Messaggio

lParam
[in] Parametro addizionale specifico e dipendente dal Messaggio

Valore Restituito

SendMessage
Il valore di ritorno restituisce il risultato dell'elaborazione e dipende dal messaggio inviato.

Link completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/messagesandmessagequeues/messagesandmessagequeuesreference/messagesandmessagequeuesfunctions/sendmessage.asp

Link abbreviato:
http://tinyurl.com/38aob

PostMessage
Se la funzione ha successo restituisce un valore <>0.
Se fallisce restituisce 0.

Link completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/messagesandmessagequeues/messagesandmessagequeuesreference/messagesandmessagequeuesfunctions/postmessage.asp

Link abbreviato:
http://tinyurl.com/vy31

ESEMPIO

Questo demo mostra come aprire un'applicazione esterna(Notepad in questo caso) e forzarne la chiusura tramite SendMessage dopo averne recuperato l'Handle tramite l'uso di FindWindow passando il ClassName dell'applicazione alla chiamata.

Per questo DEMO sono necessari 1 controllo CommandButton sul quale gestiremo l'evento Click():

CMDOPEN Come Caption del CommandButton Metteremo "*Open Notepad*"

Scrivete questo codice nel modulo di classe della Vostra Form:

```
Private Declare Function SendMessage Lib "user32" _
                         Alias "SendMessageA" _
                         (ByVal hwnd           As Long, _
                         ByVal wMsg            As Long, _
                         ByVal wParam          As Long, _
                         ByVal lParam          As Long) As Long

Private Declare Function FindWindow Lib "user32" _
                         Alias "FindWindowA" _
                         (ByVal lpClassName    As String, _
                         ByVal lpWindowName    As String) As Long

Private Const WM_CLOSE = &H10

Private Sub cmdopen_Click()
   If Me!cmdopen.Caption = "Open Notepad" Then
      Dim lNotepadHwnd As Long
      Shell "notepad.exe", vbNormalNoFocus
      Me!cmdopen.Caption = "Close Notepad"
   ElseIf Me!cmdopen.Caption = "Close Notepad" Then
      ' Si potrebbe usare anche POSTMESSAGE come mostrato sotto
      ' sostanzialmente la differenza tra le 2 API è che una attende
      ' l'esecuzione del comando inviato(SENDMESSAGE) l'altra accoda il
      ' messaggio alla coda messaggi della finestra e ritorna(POSTMESSAGE)
      'PostMessage FindWindow("Notepad", vbNullString), WM_CLOSE, 0, 0
      SendMessage FindWindow("Notepad", vbNullString), WM_CLOSE, 0, 0
      Me!cmdopen.Caption = "Open Notepad"
   End If
End Sub
```

PRIMO DEMO API

Questo Demo utilizza le API fino ad ora prese in considerazione e si prefigge di evidenziare come accedere ad una Applicazione esterna(vista come Finestra tramite il suo Handle) e di poterne manipolare le sue proprietà.
Come Applicazione esterna per il nostro test useremo NOTEPAD sia per la semplicità che offre sia purché è largamente diffuso ed utile.

Per questo DEMO sono necessari 6 controlli CommandButton sui quali gestiremo l'evento Click():

CMDOPEN_NOTEPAD
CMDSENDTEXT
CMDSENDEDIT
CMDGETTEXT
CMDGETEDIT
CMDGETLINES

ed un controllo TextBox:

txtNotepad

Scrivete questo codice nel modulo di classe della Vostra Form:

```
Private Declare Function FindWindow Lib "user32" _
                         Alias "FindWindowA" _
                         (ByVal lpClassName    As String, _
                         ByVal lpWindowName    As String) As Long

Private Declare Function FindWindowEx Lib "user32" _
                         Alias "FindWindowExA" _
                         (ByVal hWnd1          As Long, _
                         ByVal hWnd2           As Long, _
                         ByVal lpsz1           As String, _
                         ByVal lpsz2           As String) As Long

Private Declare Function SendMessageSTRING Lib "user32" _
                         Alias "SendMessageA" _
                         (ByVal hwnd           As Long, _
                         ByVal wMsg            As Long, _
                         ByVal wParam          As Long, _
                         ByVal lParam          As String) As Long

Private Declare Function SendMessageLONG Lib "user32" _
                         Alias "SendMessageA" _
                         (ByVal hwnd           As Long, _
                         ByVal wMsg            As Long, _
                         ByVal wParam          As Long, _
                         ByVal lParam          As Long) As Long

Private Const WM_GETTEXT = &HD
Private Const WM_SETTEXT = &HC
Private Const EM_GETLINECOUNT = &HBA

' La prima cosa da fare per questo test è aprire un'istanza di NOTEPAD
' in modo tale da verificare il funzionamento delle chiamate.
Private Sub CMDOPEN_NOTEPAD_Click()
    Shell "notepad.exe"
End Sub
```

```
Private Sub CMDSENDTEXT_Click()
     ‘ Questa funzione modifica il Nome o Titolo della Finestra(la CAPTION)

     Dim lNotepadHwnd    As Long           ‘ Handle di NOTEPAD
     Dim sCaption        As String * 256   ‘ Buffer stringa

     ‘ Ricaviamo l’Handle usando il ClassName di Notepad
     lNotepadHwnd = FindWindow("Notepad", vbNullString)
     sCaption = InputBox("Che nome vuoi dare alla Caption di NOTEPAD ?")

     ‘ Sfruttiamo WM_SETTEXT per indicare alla Finestra di Notepad che vogliamo
     ‘ impostare il Testo del Titolo della stessa.
     SendMessageSTRING lNotepadHwnd, WM_SETTEXT, 256, sCaption
End Sub

Private Sub CMDGETTEXT_Click()
     ‘ Questa funzione recupera il Nome della Finestra(CAPTION)

     ‘ Handle di NOTEPAD
     Dim lNotepadHwnd    As Long
     ‘ Handle della TEXTBOX(Child)
     Dim lNotepadEdit    As Long
     ‘ Buffer stringa
     Dim sCaption        As String * 256

     ‘ Ricaviamo l’Handle usando il ClassName di Notepad
     lNotepadHwnd = FindWindow("Notepad", vbNullString)

     ‘ Sfruttiamo WM_GETTEXT per indicare alla Finestra di Notepad che vogliamo
     ‘ leggere il Testo del Titolo.
     SendMessageSTRING lNotepadHwnd, WM_GETTEXT, 256, sCaption
     MsgBox sCaption
End Sub

Private Sub CMDSENDEDIT_Click()
     ‘ Questa funzione scrive nella finestra Child di NOTEPAD in questo caso è
     ‘ la finestra di editazione “TEXTBOX”

     ‘ Handle di NOTEPAD
     Dim lNotepadHwnd    As Long
     ‘ Handle della TEXTBOX(Child)
     Dim lNotepadEdit    As Long
     ‘ Buffer stringa
     Dim sCaption        As String * 256

     ‘ Ricaviamo l’Handle usando il ClassName di Notepad
     lNotepadHwnd = FindWindow("Notepad", vbNullString)

     ‘ Ricaviamo l’Handle della finestra di Editazione passando l’Handle della
     ‘ Finestra Genitore ed il ClassName della Finestra Child.
     lNotepadEdit = FindWindowEx(lNotepadHwnd, 0&, "Edit", vbNullString)
     sCaption = InputBox("Cosa vuoi scrivere in NOTEPAD ?")
     SendMessageSTRING lNotepadEdit, WM_SETTEXT, 256, sCaption
End Sub
```

```
Private Sub CMDGETEDIT_Click()
     ‘ Questa funzione recupera il contenuto della finestra Child di NOTEPAD
     ‘ in questo caso è la finestra di editazione “TEXTBOX” che ha come
     ‘ ClassName=”Edit”

     ‘ Handle di NOTEPAD
     Dim lNotepadHwnd   As Long
     ‘ Handle della TEXTBOX(Child)
     Dim lNotepadEdit   As Long
     ‘ Buffer stringa
     Dim sCaption       As String * 256

     ‘ Ricaviamo l’Handle usando il ClassName di Notepad
     lNotepadHwnd = FindWindow("Notepad", vbNullString)

     ‘ Ricaviamo l’Handle della finestra di Editazione passando l’Handle della
     ‘ Finestra Genitore ed il ClassName della Finestra Child.
     lNotepadEdit = FindWindowEx(lNotepadHwnd, 0&, "Edit", vbNullString)
     SendMessageSTRING lNotepadEdit, WM_GETTEXT, 256, sCaption
     MsgBox sCaption
End Sub

Private Sub CMDGETLINES_Click()
     ‘ Questa Funzione recupera il numero di Linee di testo presenti nella
     ‘ TextBox di NOTEPAD

     ‘ Handle di NOTEPAD
     Dim lNotepadHwnd   As Long
     ‘ Handle della TEXTBOX(Child)
     Dim lNotepadEdit   As Long
     ‘ Non è un Buffer, ma un Long
     Dim lNumOfLines    As Long

     ‘ Ricaviamo l’Handle usando il ClassName di Notepad
     lNotepadHwnd = FindWindow("Notepad", vbNullString)

     ‘ Ricaviamo l’Handle della finestra di Editazione passando l’Handle della
     ‘ Finestra Genitore ed il ClassName della Finestra Child.
     lNotepadEdit = FindWindowEx(lNotepadHwnd, 0&, "Edit", vbNullString)
     lNumOfLines = SendMessageLONG(lNotepadEdit, EM_GETLINECOUNT, 0, 0)
     MsgBox "The notepad has " & lNumOfLines & " lines"
End Sub
```

GETWINDOWRECT

Restituisce le dimensioni Rettangolari(RECT) relative alla Finestra specificata. Le dimensioni sono restituite in Pixels e sono riferite dall'angolo superiore sinistro all'angolo inferiore destro.

Sintassi

```
BOOL GetWindowRect(
    HWND          hWnd,
    LPRECT        lpRect
);
```

Parametri

hWnd
[in] Handle della Finestra.

lpRect
[out] Puntatore alla variabile strutturata RECT adatta a ricevere le coordinate in Pixels.

Valore Restituito

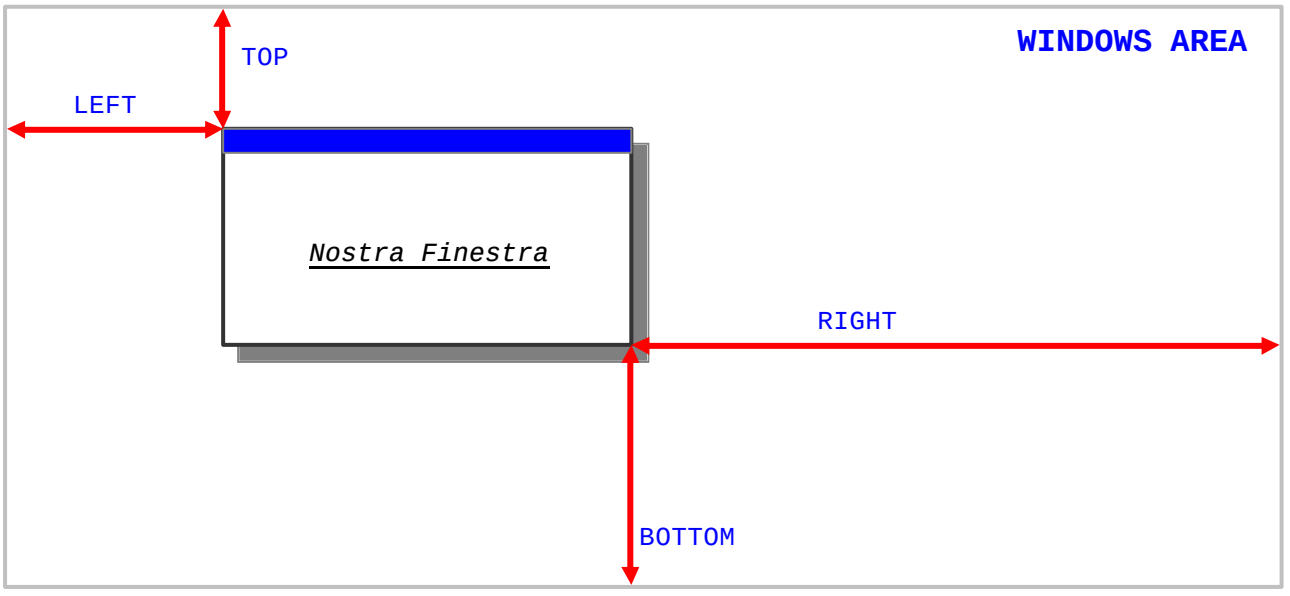
Se la funzione ha successo restituisce un valore <>0.
Se fallisce restituisce 0.

Link Completo.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/windows/windowreference/windowfunctions/getwindowrect.asp

Link abbreviato:
http://tinyurl.com/bd7eb

Struttura RECT

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



GETCLIENTRECT

Restituisce le coordinate della ClientArea relativa alla nostra Finestra, cioè l'area della finestra che non include i bordi, i menu e il titolo(quella nella quale possiamo lavorare). Le Client coordinate specificano l'angolo alto-sinistro e basso-destro della ClientArea. Poichè queste coordinate sono relative alla ClientArea stessa l'angolo alto-sinistro sarà (0,0).

Sintassi

```
BOOL GetClientRect(
    HWND          hWnd,
    LPRECT        lpRect;
```

Parametri

hWnd

[in] Handle della finestra di cui recuperare le coordinate della ClientArea.

lpRect

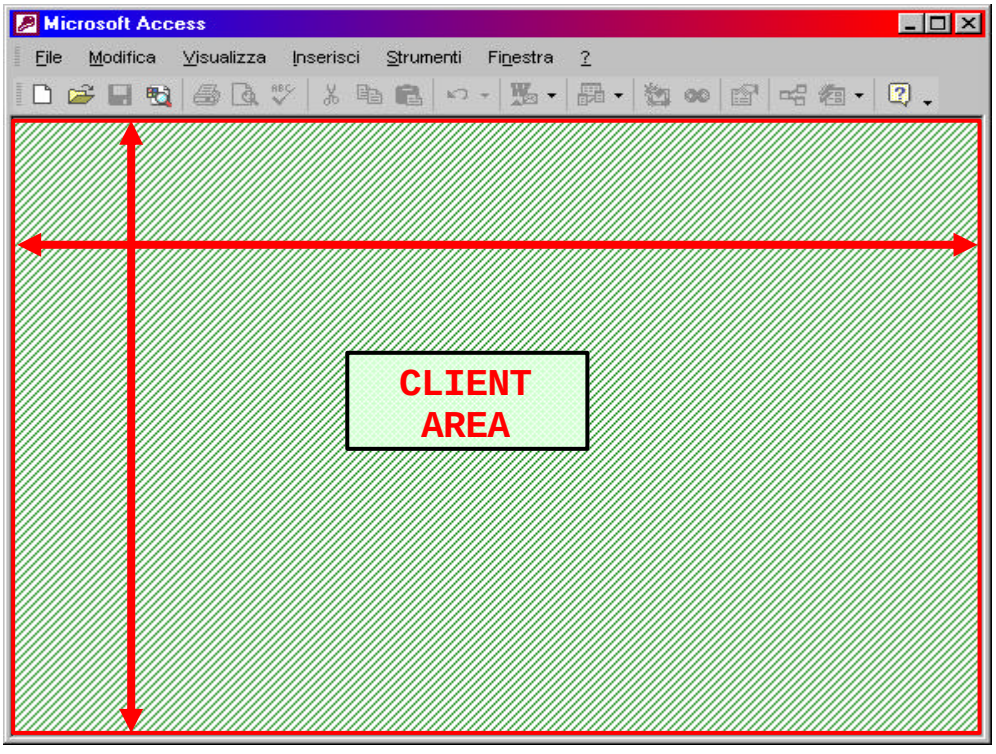
[out] Puntatore alla struttura RECT che riceverà le coordinate. I membri **left** e **top** saranno = 0 mentre **right** e **bottom** conterranno rispettivamente Larghezza(width) ed Altezza(height) della finestra.

Valore Restituito

Se la funzione ha successo restituisce un valore <>0.
Se fallisce restituisce 0.

Link Completo.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/windows/windowreference/windowfunctions/getclientrect.asp

Link abbreviato:
http://tinyurl.com/89u9v



GETPARENT

Restituisce l'Handle della finestra Parend di quella specificata.

Syntax

```
HWND GetParent(
    HWND          hWnd
```

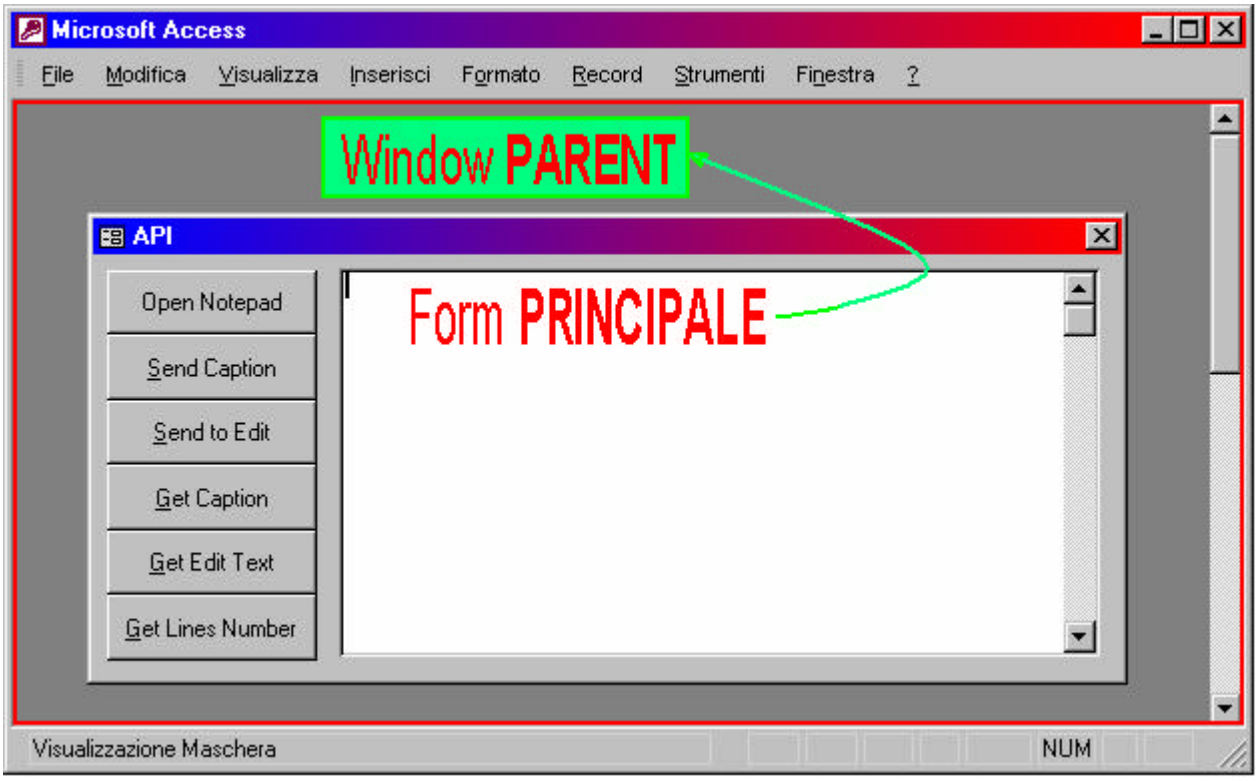
Parametri

hWnd

[in] Handle della Finestra della quale si vuole ricavare il Parent_Handle.

Valore Restituito

Se la Finestra è una Child, il valore restituito è l'Handle della Parent, se è una Finetra(tipo MDI) restituisce l'Handle di Windows. If the window is a top-level unowned window or if the function fails, the Valore Restituito is NULL. To get extended error information, call GetLastError. For example, this would determine, when the function returns NULL, if the function failed or the window was a top-level window.



Link completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/windows/windowreference/windowfunctions/getparent.asp
Link abbreviato:
http://tinyurl.com/axtuz

MOVEWINDOW

Cambia la posizione e le dimensioni della Finestra specificata. Per le Finestre Popup, la posizione e le dimesioni sono relative all'angolo superiore sinistro dello schermo. Per le finestre Child sono relativi all'angolo superiore sinistro della Parent(ClientArea).

Sintassi

```
BOOL MoveWindow(
    HWND          hWnd,
    INT           X,
    INT           Y,
    INT           nWidth,
    INT           nHeight,
    BOOL          bRepaint
);
```

Parametri

hWnd
[in] Handle della finestra.

X
[in] Nuova coordinate Left.

Y
[in] coordinate Top.

nWidth
[in] Nuova larghezza(width).

nHeight
[in] Nuova altezza(height).

bRepaint
[in] Specifica se la Finestra deve essere ridisegnata. Se questo parametro=TRUE la finestra riceve il messaggio WM_PAINT. Se il parametro= FALSE, nessun tipo di aggiornamento grafico viene eseguito.

Valore Restituito
Se la funzione ha successo restituisce un valore <>0.
Se fallisce restituisce 0.

Link Completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/windows/windowreference/windowfunctions/movewindow.asp

Link abbreviato:
http://tinyurl.com/9mvru

NOTE
MoveWindow invia questi messaggi:

WM_WINDOWPOSCHANGING
WM_WINDOWPOSCHANGED
WM_MOVE
WM_SIZE
WM_NCCALCSIZE

SECONDO DEMO API

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

Private Declare Function MoveWindow Lib "user32" _
                         (ByVal hWnd     As Long, _
                         ByVal x         As Long, _
                         ByVal y         As Long, _
                         ByVal nWidth    As Long, _
                         ByVal nHeight   As Long, _
                         ByVal bRepaint  As Long) As Long

Private Declare Function GetDesktopWindow Lib "user32" () As Long

Private Declare Function GetWindowRect Lib "user32" _
                         (ByVal hWnd     As Long, _
                         lpRect          As RECT) As Long

Private Declare Function GetParent Lib "user32" _
                         (ByVal hWnd     As Long) As Long

Public Sub Center(frm As Form)
' Centra la Maschera nell'Area della MDI parent(Access), se la Form è Popup
' allora il centraggio avviene rispetto all'area dello schermo

    Dim hWndParent       As Long
    Dim rctParent        As RECT
    Dim rct              As RECT
    Dim intWidth         As Integer
    Dim intHeight        As Integer
    Dim intParentWidth   As Integer
    Dim intParentHeight  As Integer

    On Error GoTo HandleErrors
    ' Recupera le coordinate della Form.
    Call GetWindowRect(frm.hWnd, rct)

    ' Recupera l'Handle della Finestra MDI.
    hWndParent = GetParent(frm.hWnd)
    ' Se la Form non è MDI allora hwndParent<>0 e coinciderà con l'Handle
    ' dell'applicazione, altrimenti la Form è Popup.
    If hWndParent <> Application.hWndAccessApp Then
        Call GetWindowRect(hWndParent, rctParent)
    Else
        ' Essendo POPUP recupera le coordinate del desktop.
        Call GetWindowRect(GetDesktopWindow(), rctParent)
    End If
    ' Calcola width/height della Parent (indifferentemente che sia Access MDI,
    ' oppure lo schermo(se Popup).
    intParentWidth = rctParent.Right - rctParent.Left
    intParentHeight = rctParent.Bottom - rctParent.Top
```

```
        ' Calcola width della Form, e le nuove coordinate relative alla Parent.
        With rct
             intWidth = .Right - .Left
             intHeight = .Bottom - .Top
             .Left = (intParentWidth - intWidth) \ 2
             .Top = (intParentHeight - intHeight) \ 2
        End With

        ' Sposta la Form nella nuova posizione.
        Call MoveWindow(frm.hWnd, _
                        rct.Left, _
                        rct.Top, _
                        intWidth, _
                        intHeight, _
                        bRepaint:=True)

ExitHere:
        Exit Sub

HandleErrors:
        Select Case Err.Number
             Case Else
                  Err.Raise Err.Number, Err.Source, _
                  Err.Description, Err.HelpFile, Err.HelpContext
        End Select
End Sub
```

TERZO DEMO API

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

Type COORDS
    Left                As Long
    Top                 As Long
    Width               As Long
    Height              As Long
End Type

Private Declare Function FindWindowEx Lib "user32" _
                         Alias "FindWindowExA" _
                         (ByVal hWnd1    As Long, _
                         ByVal hWnd2     As Long, _
                         ByVal lpsz1     As String, _
                         ByVal lpsz2     As String) As Long

Private Declare Function GetClientRect Lib "user32" _
                         (ByVal hwnd     As Long, _
                         lpRect          As RECT) As Long

Private Declare Function MoveWindow Lib "user32" _
                         (ByVal hWnd     As Long, _
                         ByVal x         As Long, _
                         ByVal y         As Long, _
                         ByVal nWidth    As Long, _
                         ByVal nHeight   As Long, _
                         ByVal bRepaint  As Long) As Long

' Questa funzione ridimensiona la Form passata calcolando la Client area della
' MDI_ACCESS sfruttando il ClassName
Public Sub FillMDIClientArea(frm As Access.Form)
     Dim mC              As COORDS
     Dim lpRect          As RECT
     Dim hwndCli         As Long
     ' Ricaviamo l'Handle della CLIENT Area
     hwndCli = FindWindowEx(Access.hWndAccessApp, 0, "MDIClient", vbNullString)
     If hwndCli = 0 Then Exit Sub
     GetClientRect hwndCli, lpRect
     ' Convertiamo le dimensioni Rettangolari in Coordinate
     mC = RctToCoords(lpRect)
     Call MoveWindow(frm.hwnd, 0, 0, mC.Width, mC.Height, 1)
End Sub

Public Function RctToCoords(rct As RECT) As COORDS
     ' Converte da RECT a COORDS
     Dim c               As COORDS
     With c
          .Left = rct.Left
          .Top = rct.Top
          .Width = rct.Right - rct.Left
          .Height = rct.Bottom - rct.Top
     End With
     RctToCoords = c
End Function
```

SHOWWINDOW

Imposta un determinato stato di visualizzazione della Finestra.

Sintassi

```
BOOL ShowWindow(
    HWND          hWnd,
    INT           nCmdShow
);
```

Parametri

hWnd

[in] Handle della Finestra.

nCmdShow

[in] Specifica come la Finestra deve essere visualizzata. Questo parametro è ignorato la prima volta che l'applicazione chiama **ShowWindow**, se il programma che ha lanciato l'applicazione ha interrogato la struttura STARTUPINFO. Altrimenti, la prima chiamata a **ShowWindow**, il valore sarà quello ottenuto dal parametro *nCmdShow* della funzione WinMain. Nelle successive chiamate, questo parametro può assumere uno dei seguenti valori:

SW_FORCEMINIMIZE
Windows 2000/XP: Minimizza la finestra anche se il thread che occupa non risponde.

SW_HIDE
Nasconde la Finestra attivandone un'altra.

SW_MAXIMIZE
Massimizza la finestra.

SW_MINIMIZE
Minimizza la finestra e attiva la finestra Popup successiva in ordine progressivo(Z).

SW_RESTORE
Attiva e visualizza la finestra. Se è Minimizzata o Massimizzata, il sistema la ripristina alle dimensioni ed alla posizione originali.

SW_SHOW
Attiva e visualizza la finestra nelle sue attuali coordinate e dimensioni.

SW_SHOWDEFAULT
Imposta la visibilità in base alla struttura **STARTUPINFO** passata con l'uso della funzione CreateProcess attraverso il programma che ha lanciato l'applicazione.

SW_SHOWMAXIMIZED
Attiva, visualizza e Massimizza la finestra.

SW_SHOWMINIMIZED
Attiva, visualizza e Minimizza la finestra.

SW_SHOWMINNOACTIVE
Visualizza e Minimizza la finestra senza attivarla

SW_SHOWNA
Visualizza la finestra nelle sue attuali coordinate e dimensioni senza attivarla.

SW_SHOWNOACTIVATE
Mostra la finestra nelle sue più recenti dimensioni e posizione. Questo valore è simile a SW_SHOWNORMAL, ad eccezione che la finestra non viene attivata.

SW_SHOWNORMAL
Attiva e visualizza la finestra. Se è Minimizzata o Massimizzata, il sistema la ripristina alle dimensioni ed alla posizione originali. Un'applicazione deve specificare questo Flag alla prima chiamata.

Valore Restituito
Se la funzione ha successo restituisce un valore <>0.
Se fallisce restituisce 0.

Link completo
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/windows/windowreference/windowfunctions/showwindow.asp

Link abbreviato
http://tinyurl.com/cohx

ESEMPIO

```
Declare Function ShowWindow Lib "user32" _
                         Alias "ShowWindow" _
                         (ByVal hwnd           As Long, _
                         ByVal nCmdShow        As Long) As Long

Const SW_HIDE = 0
Const SW_SHOWNORMAL = 1
Const SW_SHOWMINIMIZED = 2
Const SW_SHOWMAXIMIZED = 3
Const SW_SHOWNOACTIVE = 4
Const SW_SHOW = 5
Const SW_MINIMIZE = 6
Const SW_SHOWMINNOACTIVE = 7
Const SW_SHOWNA = 8
Const SW_RESTORE = 9

Function MaximizeApp()
   Dim Maxit      As Long
   Maxit = ShowWindow(hWndAccessApp, SW_SHOWMAXIMIZED)
End Function

Function MinimizeApp()
   Dim Minit      As Long
   Minit = ShowWindow(hWndAccessApp, SW_SHOWMINIMIZED)
End Function

Function RestoreApp()
   Dim Restoreit  As Long
   Restoreit = ShowWindow(hWndAccessApp, SW_SHOWNORMAL)
End Function
```

Questi esempi non sono necessari se l'azione viene eseguita sul nostro applicativo VBA, ma solo se sono indirizzati ad applicative esterni, poichè in VBA sono disponibili queste funzioni Native:

```
Application.DoCmd.RunCommand acCmdAppMaximize
Application.DoCmd.RunCommand acCmdAppMinimize
Application.DoCmd.RunCommand acCmdAppRestore
```

ISWINDOW

Determina se una Finestra esiste attraverso il suo identificativo(Handle).

Sintassi

```
BOOL IsWindow(
    HWND          hWnd
);
```

Parametri

hWnd
[in] Handle della finestra da testare.

Valore Restituito
Se l'Handle identifica una finestra esistente, il valore restituito <>0.
Se l'Handle identifica una finestra non esistente, il valore restiutuito=0.

```
Declare Function IsWindow Lib "user32" _
                          Alias "IsWindow" _
                          (ByVal hwnd As Long) As Long
```

Link Completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/windows/windowreference/windowfunctions/iswindow.asp

Link abbreviato:
http://tinyurl.com/d33u6

ISWINDOWVISIBLE

Restituisce lo stato di Visibilità della Finestra specificata.

Sintassi

```
BOOL IsWindowVisible(
    HWND          hWnd
);
```

Parametri
hWnd
[in] Handle della Finestra da testare.

Valore Restituito
Se la Finestra specificata, la sua Parent e la Parent della Parent ecc… hanno lo stile WS_VISIBLE style restituisce un valore diverso da zero, altrimenti il valore è zero.
Poiché il valore restituito specifica in ogni caso lo stato WS_VISIBLE è possibile che il valore sia diverso da Zero anche se un'altra finestra copre la nostra.

Link Completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/windows/windowreference/windowfunctions/iswindowvisible.asp

Link abbreviato:
http://tinyurl.com/c39m9

ESEMPIO

```
Declare Function IsWindowVisible Lib "user32.dll" _
                          (ByVal hwnd As Long) As Long

Private Sub Form_Activate()
    'Is window visible?
    MsgBox IsWindowVisible(Me.hwnd)
End Sub

Private Sub Form_Load()
    'Is window visible?
    MsgBox IsWindowVisible(Me.hwnd)
End Sub
```

SHELLEXECUTE

Esegue un'operazione su un file specifico.

Sintassi

```
HINSTANCE ShellExecute(
    HWND          hwnd,
    LPCTSTR       lpOperation,
    LPCTSTR       lpFile,
    LPCTSTR       lpParametri,
    LPCTSTR       lpDirectory,
    INT           nShowCmd
);
```

Parametri

hwnd

[in] Handle della finestra associata alla visualizzazione dell'interfaccia. Questo valore può essere nullo(0) se l'operazione non è associata a nessuna finestra.

lpOperation

[in] Puntatore ad una stringa a terminazione nulla che specifica l'azione da effettuare. I comandi o azioni sono generalmente dipendenti dal tipo di oggetto. Per maggiori dettagli vedere Object Verbs. Di seguito i comandi più usati:

"**EDIT**"
Apre un editor ed il relativo file in modalità modifica. Se *lpFile* non è un file la funzione fallisce.

"**EXPLORE**"
Esplora la cartella specificata da *lpFile*.

"**FIND**"
Inizia la ricerca a partire dalla directory specificata.

"**OPEN**"
Apre il file specificato dal parametro *lpFile*. Il file può essere un EXE, un documento o una cartella(directory).

"**PRINT**"
Stampa il documento specificato dal parametro *lpFile*. Se *lpFile* non è un file la funzione fallisce.

"**NULL**"
Per i sistemi precedenti a Microsoft Windows 2000, il commando di Default è usato se presente nel Registry. Se non presente viene usato "OPEN".
Per Windows 2000 e successivi, se il commando di Default è disponibile nel Registry. Se non presente viene usato "OPEN". Se entrambi non sono disponibili il sistema usa il primo nella lista dei comandi presenti nel Registry.

lpFile

[in] Puntatore ad una stringa a terminazione nulla che specifica il file o l'oggetto nel quale eseguire il comando specifico. Notare che non tutti i comandi sono supportati da tutti gli oggetti. Ad esempio non tutti i tipi di documenti supportano il commando "PRINT".

lpParametri

[in] Se il parametro *lpFile* specifica un EXE, *lpParametri* è un puntatore ad una stringa a terminazione nulla anche specifica i parametri da passare all'applicazione. Il formato di questa stringa è determinato dal verbo invocato. Se *lpFile* specifica un file , *lpParametri* può essere NULL.

lpDirectory
[in] Puntatore ad una stringa a terminazione nulla che identifica la Directory di Default.

nShowCmd
[in] Flag che specifica come un'applicazione viene visualizzata quando viene aperta. Se *lpFile* specifica un file il Flag è passato all'applicazione EXE associata.

SW_HIDE
Nasconde la Finestra attivandone un'altra.
SW_MAXIMIZE
Massimizza la finestra.
SW_MINIMIZE
Minimizza la finestra e attiva la finestra Popup successiva in ordine progressivo(Z).
SW_RESTORE
Attiva e visualizza la finestra. Se è Minimizzata o Massimizzata, il sistema la ripristina alle dimensioni ed alla posizione originali.
SW_SHOW
Attiva e visualizza la finestra nelle sue attuali coordinate e dimensioni.
SW_SHOWDEFAULT
Imposta lo stato di visualizzazione basato sul valore SW_ specificato nella struttura **STARTUPINFO** passata dalla funzione CreateProcess dal programma che lancia l'applicativo.
SW_SHOWMAXIMIZED
Attiva, visualizza e Massimizza la finestra.
SW_SHOWMINIMIZED
Attiva, visualizza e Minimizza la finestra.
SW_SHOWMINNOACTIVE
Visualizza e Minimizza la finestra senza attivarla
SW_SHOWNA
Visualizza la finestra nelle sue attuali coordinate e dimensioni senza attivarla.
SW_SHOWNOACTIVATE
Mostra la finestra nelle sue più recenti dimensioni e posizione. Questo valore è simile a SW_SHOWNORMAL, ad eccezione che la finestra non viene attivata.
SW_SHOWNORMAL
Attiva e visualizza la finestra. Se è Minimizzata o Massimizzata, il sistema la ripristina alle dimensioni ed alla posizione originali. Un'applicazione deve specificare questo Flag nella prima chiamata.

Valore Restituito
Il valore restituito è > 32 se la funzione ha recuperato correttamente l'eseguibile, viceversa il valore sarà <=32 se si è verificato un errore.

Link Completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/shellcc/platform/shell/reference/functions/shellexecute.asp

Link abbreviato:
http://tinyurl.com/hsgz

Vedi:
http://www.mvps.org/access/api/api0018.htm

ESEMPIO

```
Public Declare Function ShellExecute Lib "shell32.dll" _
                        Alias "ShellExecuteA" ( _
                        ByVal hwnd            As Long, _
                        ByVal lpOperation     As String, _
                        ByVal lpFile          As String, _
                        ByVal lpParametri     As String, _
                        ByVal lpDirectory     As String, _
                        ByVal nShowCmd        As Long) As Long

Public Const SW_SHOWNORMAL=1

Aprire un file NOTEPAD
Sub ShellExec()
     Dim strFile          As String
     Dim lngErr           As Long

     strFile = "c:\somefile.txt"
     lngErr = ShellExecute(0, "OPEN", strFile, "", "", SW_SHOWNORMAL)
End Sub

Aprire il Browser di posta per inviare una Mail
Sub ShellMailTo()
     Dim strFile          As String
     Dim lngErr           As Long

     strFile = "mailto:ik2zok@libero.it?subject=Test API&body=Testo da inviare"
     lngErr =  ShellExecute(0, "OPEN", strFile, "", "",  SW_SHOWNORMAL)
End Sub

Manda in stampa un file
Sub ShellPrintTo()
     Dim strFile          As String
     Dim lngErr           As Long

     strFile = "C:\MyFolder\MyDocument.Doc"
     lngErr =  ShellExecute(0, "PRINT", strFile, "", "",  SW_SHOWNORMAL)
End Sub

Esegue un file Audio
Sub ShellPrintTo()
     Dim strFile          As String
     Dim lngErr           As Long

     strFile = "C:\Music\Mozart.Wav"
     lngErr =  ShellExecute(0, "PLAY", strFile, "", "",  SW_SHOWNORMAL)
End Sub

Apre la cartella proprietà del file Link
Sub ShellPrintTo()
     Dim strFile          As String
     Dim lngErr           As Long

     strFile = "C:\Windows\Desktop\Word.Lnk"
     lngErr =  ShellExecute(0, "PROPERTIES", strFile, "", "",  SW_SHOWNORMAL)
End Sub
```

SHELLEXECUTEEX

Gestisce le azioni sull'esecuzione di un File.

Syntax

```
BOOL ShellExecuteEx(
    LPSHELLEXECUTEINFO   lpExecInfo
);
```

Parametri

lpExecInfo

Indirizzo della struttura SHELLEXECUTEINFO che contiene e riceve le informazioni relative all'applicazione che deve essere eseguita.

Valore Restituito

Restituisce TRUE eseguita correttamente, FALSE in altro caso.

```
Declare Function ShellExecuteEx Lib "shell32.dll" _
                        Alias "ShellExecuteExA" _
                        (lpExecInfo As SHELLEXECUTEINFO) As Long
```

Struttura SHELLEXECUTEINFO

```
Private Type SHELLEXECUTEINFO
    cbSize              As Long
    fMask               As Long
    hwnd                As Long
    lpVerb              As String
    lpFile              As String
    lpParametri         As String
    lpDirectory         As String
    nShow               As Long
    hInstApp            As Long
    lpIDList            As Long     'Optional
    lpClass             As String   'Optional
    hkeyClass           As Long     'Optional
    dwHotKey            As Long     'Optional
    hIcon               As Long     'Optional
    hProcess            As Long     'Optional
End Type
```

Link completo:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/shellcc/platform/shell/reference/functions/shellexecuteex.asp

Link abbreviato:
http://tinyurl.com/5fucd

ESEMPIO
http://vbnet.mvps.org/index.html?code/shell/propertypage.htm

SETCURSORPOS

Muove il cursore alle specifiche coordinate di schermo(Pixels

Sintassi

```
BOOL SetCursorPos(
    int X,
    int Y;
```

Parametri

X

[in] Specifica la nuova coordinata sull'asse X(in Pixels).

Y

[in] Specifica la nuova coordinata sull'asse Y(in Pixels).

Return Value

Restituisce un valore NonZero in caso di successo altrimenti Zero

```
Declare Function SetCursorPos Lib "user32" _
                         Alias "SetCursorPos" _
                         (ByVal x As Long, _
                         ByVal y As Long) As Long
```

E' possibile per comodità(vedremo in seguito l'applicazione) usare una variabile strutturata tipo POINTAPI:

Struttura POINTAPI

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

La funzione SetCursorPos sposta il cursore del mouse in una determinata posizione dello schermo.

ESEMPIO

L'esempio di codice seguente, mostra come posizionare al centro del monitor, il cursore del mouse(supponiamo una risoluzione video 1024x768):

```
Sub SpostaCenterScreen()
    Dim Posizione As POINTAPI
    Posizione.x = 1024\ 2    'metà larghezza dello schermo in pixel
    Posizione.y = 768 \ 2    'metà altezza dello schermo in pixel
    'sposta il cursore del mouse al centro dello schermo
    SetCursorPos Posizione.x, Posizione.y
End Sub
```

La funzione SetCursorPos sposta la posizione del mouse rispetto all'angolo superiore sinistro dello schermo. Se necessita invece spostare il cursore, relativamente alla finestra in uso (form) è necessario anche l'uso della dichiarazione API ClientToScreen

CLIENTTOSCREEN

La funzione *ClientToScreen* converte i valori impostati con la precedente funzione, *SetCursorPos*, ad una determinata finestra e non allo schermo.

```
BOOL ClientToScreen(
  HWND              hWnd,        // handle to window
  LPPOINT           lpPoint      // screen coordinates
```

Parametri

hWnd

[in] Handle to the window whose client area is used for the conversion.

lpPoint

[in/out] Pointer to a **POINT** structure that contains the client coordinates to be converted. The new screen coordinates are copied into this structure if the function succeeds.

```
Declare Function ClientToScreen Lib "user32" _
                         Alias "ClientToScreen" _
                         (ByVal hwnd As Long, _
                         lpPoint As POINTAPI) As Long
```

ESEMPIO

L'esempio di codice seguente, mostra come posizionare al centro di una finestra (form), il cursore del mouse:

```
Sub SpostaCenterWindow()
    Dim Posizione As POINTAPI
    'metà larghezza della finestra in pixels
    Posizione.x = fTwips2Pixels(Me.Width \ 2, 0)
    'metà altezza della finestra in pixel
    Posizione.y = fTwips2Pixels (Me.Height \ 2,1)
    'finestra od oggetto di riferimento per la funzione SetCursorPos
    ClientToScreen Me.hWnd, Posizione
    'sposta il cursore del mouse al centro della finestra
    SetCursorPos Posizione.x, Posizione.y
End Sub
```

La funzione fTwips2Pixels converte i Twips passati in coordinate di schermo(Pixels) la trovate nel mio sito:

http://www.mantuanet.it/alessandro.baraldi/
Sezione [**API**]
1.22 Conversione Twips/Pixels e viceversa

Allo stesso modo, cambiando i riferimenti, è possibile spostare il cursore del mouse al centro di un'altro qualsiasi controllo, come ad esempio, un CommandButton, ricordando sempre che in Access un controllo espone l'Handle solo quando ha lo stato attivo.

WINDOWFROMPOINT

La funzione WindowFromPoint restituisce l'handle (hWnd) della finestra che si trova in corrispondenza delle coordinate x,y specificate nell'argomento usato per la chiamata.

Sintassi

```
HWND WindowFromPoint(
    POINT Point);
```

Parametri

Point

[in] Specifica la struttura POINT che definisce le coordinate da usare.

Return Value

The return value is a handle to the window that contains the point. If no window exists at the given point, the return value is NULL. If the point is over a static text control, the return value is a handle to the window under the static text control.

GETCURSORPOS

La funzione GetCursorPos restituisce, in una variabile di tipo definito dall'utente denominata POINTAPI, le coordinate x,y della posizione del mouse, relative all'angolo superiore sinistro dello schermo. I valori restituiti sono espressi in pixels.

Sintassi

```
BOOL GetCursorPos(
    LPPOINT lpPoint);
```

Parametri

lpPoint

[out] Puntatore ad una struttura tipo POINT che riceve le coordinate in Pixels della posizione del cursore.

Valore restituito

Restituisce un valore NonZero in caso di successo altrimenti Zero

ESEMPIO 1

Per identificare la finestra o l'oggetto che si trova, in un determinato momento, sotto al cursore del mouse, è necessario usare le seguenti funzioni API:

```
Declare Function GetCursorPos Lib "user32" _
                         Alias "GetCursorPos" (lpPoint As POINTAPI) As Long

Declare Function WindowFromPoint Lib "user32" _
                         (ByVal xPoint As Long, _
                         ByVal yPoint As Long) As Long
```

Utilizzando la seguente variabile del tipo POINTAPI:

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

L'esempio di codice seguente, mostra come utilizzare la funzione:

```
Function WindowHandle() As Long
     Dim Pos As POINTAPI
     'restituisce la posizione x,y del cursore del mouse (in pixel)
     GetCursorPos Pos
     'ricavo l'handle della finestra che si trova in corrispondenza di: Pos.x,
     Pos.y
     WindowHandle = WindowFromPoint(Pos.x, Pos.y)
End Function
```

La funzione WindowHandle restituirà l'handle (hWnd) di qualunque finestra (anche se questa non appartiene alla propria applicazione) che si trova sotto al cursore del mouse.

ESEMPIO 2

Il seguente codice di esempio, permette di ricavare tutti gli handle degli oggetti presenti nel vostro desktop:

```
'proprietà Timerinterval = 500
Private Sub Form_Timer()
     'visualizza l'handle dell'oggetto sopra a cui si trova il cursore del mouse
     Label1. Caption = WindowHandle
End Sub

Function WindowHandle() As Long
     Dim Pos As POINTAPI
     'restituisce la posizione x,y del cursore del mouse (in pixel)
     GetCursorPos Pos
     WindowHandle = WindowFromPoint(Pos.x, Pos.y)
End Function
```

DOCUMENTAZIONE UTILE

La documentazione sulle API per VBA è assente, ma in realtà non è un problema, approfitteremo di tutto quello che il WEB mette a disposizione relativamente al linguaggio VB6 per le API a 32Bit.
Prima di tutto è utile e fondamentale avere uno strumento che vi agevola con la ricerca, il raggruppamento per argomentazioni, la spiegazione dettagliata dei parametri, le dichiarazioni corrette e cosa non da meno esempi pratici e semplici di come accedere alle API in questione.
Questo strumento è un file che potete scaricare dal sito sottolinkato, purtroppo non è aggiornatissimo, ma direi che soddisfa senza problemi il 90% delle normali esigenze di chi inizia ed anche oltre:

API-GUIDE
http://www.mentalis.org/agnet/apiguide.shtml

Il sito di Mentalis(EX Allapi) espone anche ottimi tutorial vari sulla programmazione avanzata in VB, facilmente riconducibile al VBA con un minimo di attenzione.

http://www.mentalis.org/tips/tips.shtml

Importantissimo Link al sito Microsoft:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winprog/winprog/windows_api_start_page.asp

Un sito estremamente ricco di documentazione(anche se per VB6) è quello di Randy Birch, consiglio di non perderlo:

http://vbnet.mvps.org/index.html

Anche se un pò datato e non aggiornatissimo rimane un punto di riferimento il sito di Dev Ashish's

http://www.mvps.org/access/

Altri:
http://www.mvps.org/vbvision/
http://www.planet-source-code.com/vb/default.asp?lngWId=1
http://www.vbfrance.com/
http://www.vbaccelerator.com/home/index.asp