Introduzione (e qualcosa in più) alle API di Windows in Visual Basic

Prima di iniziare.

Questo articolo non intende essere sostitutivo dei manuali specifici, quali il SDK della Microsoft, ma solo facilitare l'apprendimento delle API Win32 in modo semplice, con un occhio di riguardo ai principianti e - perché no ? - anche a quelli che già da tempo le usano, mostrando il punto di vista di chi si avvicina alla loro implementazione nei programmi, diversa da quella che in genere predilige esporre gli argomenti in maniera pressoché scontata.

Occorre all'inizio cercare nel menu di Visual Basic **Strumenti\Opzioni** le voci **Salva con conferma** e **Dichiarazioni di variabili obbligatoria**, di conseguenza settarle di modo che l'istruzione **Option Explicit** appaia in tutti gli script e ad ogni esecuzione del vostro programma vi verrà chiesto se desiderate salvare il vostro lavoro (e voi lo farete per sicurezza).

Poi tenete a portata di mano il **Visualizzatore testo API** presente nella cartella di VB.

Visione d'insieme.

Avviate il Visualizzatore testo API e dal menu **File\Carica file di testo...** caricate il file **Win32api.txt**, di seguito eseguite dal menu **File\Converti testo in database...** seguendo le istruzioni e caricate quest'ultimo.

In esso vi sono tre ListBox (Tipo API, Elementi disponibili e Elementi selezionati) e tre Pulsanti (Aggiungi, Rimuovi e Copia). Se cliccate su una delle voci in *Elementi disponibili* il pulsante *Aggiungi* si attiva e dopo averlo premuto apparirà nelle voci di *Elementi selezionati*: potrete rimuoverla (*Rimuovi*) dopo averla selezionata oppure copiarla (*Copia*) negli appunti, da dove sarà possibile inserirla nel vostro progetto VB con *Incolla*.

Questo piccolo programmino fornito nel pacchetto serve a facilitare la digitazione delle dichiarazioni API desiderate, poiché, essendo alcune piuttosto lunghette, ci si può sbagliare con facilità.

Che cosa sono le API

Esse sono delle funzioni che possono a volte non avere un valore di ritorno; le stesse assomigliano in un certo qual modo alle chiamate DOS/BIOS che qualche volta avrete visto nei programmi assembly o inline in quelli C++ o TPascal: di fatto esse sono le funzioni di sistema che di solito vengono usate in Visual C++, Borland C++, Watcom C++, Visual Age for C++, etc. È il Sistema Operativo che dialoga con noi offrendoci tutto ciò che esso ci mostra: le finestre, i movimenti, i suoni la grafica, il controllo del mouse, etc.

Adesso s'inizia

Chiarito questo, se avrete lasciato la voce *Dichiarazioni* nella ListBox *Tipo API*, dopo aver cercato la voce GetWindowsDirectory e averla aggiunta in quella in basso, potrete notare che essa è composta da più parti:

(è una funzione API che ci restituisce la directory effettiva in cui è situato Windows)

- Questa è la funzione richiesta

```
Declare Function GetWindowsDirectory Lib "kernel32" Alias "GetWindowsDirectoryA" (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

- Qui di seguito la spiegazione

`Declare Function` è la **dichiarazione** vera e propria

`GetWindowsDirectory Lib "kernel32"` il **nome della funzione** e **la libreria d'origine**

`Alias "GetWindowsDirectoryA"` l'**alias**, cioè il nome sostitutivo, dato che a volte può essere identico a una istruzione VB, noterete che essa termina con una 'A', essa sta ad indicare **ANSI** , cioè set dei caratteri ad un solo byte, usato per il Win32 da Windows95. Mentre se andrete a spulciare il file Win32api.txt sotto Windows NT ne troverete un altro un altro col suffisso 'W' che sta ad indicare **WIDE**, esteso, cioè appartiene al set dei caratteri **Unicode** di questo SO. Di questo se ne preoccupa sia il compilatore, che il programma di installazione VB, ma voi dovete sapere che il VB utilizza quello di Windows NT ed è costretto ad operare la conversione sotto W95.

`(ByVal lpBuffer As String, ByVal nSize As Long)` sono gli **argomenti.** La prima è una stringa terminante con zero che serve da buffer (in realtà un vettore), mentre la seconda è la grandezza della stringa stessa, di cui il SDK ci informa che non deve superare 144. Ora poiché in C/C++ i vettori (vengono usati per formare stringhe, quest'ultime non esistono in C/C++) hanno gli indici che iniziano con zero (0), noi la dovremo inizializzare a 145, perciò dovremo con un accorgimento adattare la nostra stringa a al vettore del C/C++ e di seguito prelevare il testo sottraendolo dagli zeri successivi. Ambedue sono argomenti che devono essere passati per valore (byVal), cioè la nostra stringa conterrà il valore cercato.

`As Long` questo è il valore di ritorno, di cui il SDK ci informa che è la grandezza, in bytes, della stringa (meglio vettore) contenuta in *lpbuffer*, mentre l'eventuale errore dovrà essere prelevato tramite la funzione *GetLastError(...)*

La dichiarazione andrà messa in un modulo standard e dovrà essere richiamata dalla **sub** desiderata. Ciò che noterete più avanti è l'uso di una particolare tecnica per dimensionare una stringa di modo tale che venga usata come buffer

compatibile col tipo vettore C/C++: prima la si dichiara moltiplicandola per la grandezza desiderata, poi la si riempie di zero con la funzione interna Vb **string(…, …)**, così come si potrà notare di seguito:

```
Dim Risult&                                    ‘questo è il valore di ritorno della funzione
Dim WinDir As String * 145                     ‘dimensionamento della stringa
WinDir = String(256, 0)                        ‘ la stringa viene riempita di zero
Risult& = GetWindowsDirectory (WinDir, 145)    ‘viene chiamata
Print "valore di ritorno = " & Risult&                          ‘stampa il valore di ritorno
Print "Directory Windows = " & StrNulToStr(WinDir, Risult&)     ‘stampa directory
```

Nella seconda istruzione di stampa (print …) avrete notato due cose strane: un richiamo a una funzione di cui non si è parlato prima e il fatto che il valore di ritorno desiderato da noi, che avrebbe dovuto farci conoscere il percorso in cui è situato Windows, non sta dove ce lo saremmo aspettato, cioè nella variabile Risult&!

Poco fa abbiamo detto che il C/C++ non avendo le stringhe utilizza dei vettori di bytes per simularle e ciò in qualche modo al VB potrebbe anche star bene, in quanto comunque ha utilizzato una stringa per avere il valore, ma ciò che il VB non potrà mai fare è trattare delle stringhe in cui, oltre ai caratteri, vi sono degli zero: di fatto per lui sono un po’ indigesti. Perciò bisogna implementare una funzione che tolga gli zero e restituisca una stringa pura VB sapendo che già *Risult&* è la lunghezza di essa, cioè in tal modo:

```
' funzione che trasforma un stringa terminante con zero
' in una senza
Public Function StrNulToStr(ByVal Str_Nul$, ByVal Size As Long) As String
  Dim k, SizeKont As Long
    SizeKont = Size
   If Size = 0 Then SizeKont = 256 ' valore di default se non si passa
                                   ' alcun argomento per la lunghezza desiderata
   For k = 1 To SizeKont
     If Asc(Mid(Str_Nul$, k, 1)) = 0 Then Exit For ' per sicurezza quando vi è zero è meglio uscire
     StrNulToStr = StrNulToStr + Mid(Str_Nul$, k, 1)
   Next
End Function
```

Per ciò che riguarda il secondo dubbio bisogna chiarire che voi probabilmente siete abituati a pensare che le funzioni abbiano sempre la sintassi classica *y=f(x)*, in questo caso è sufficiente inserire il valore desiderato in **x** per avere un risultato a lui dipendente in **y**, ma con le API di Windows non funziona così: il valore y può essere il risultato dell’esito della funzione (è andata bene o male?) o qualcos’altro (in questo caso la grandezza del vettore), le funzioni dovrete sempre pensarle come le chiamate Dos/Bios in Assembly, laddove si parla di **servizi del Sistema Operativo**. Inoltre è proprio pratica del C e successivamente del C++ (anche se quest’ultimo è molto più tipizzato dello pseudo-genitore) pensare alle funzioni come un qualcosa in cui vi si può far passare, dopo aver impostato i suoi argomenti come puntatori, anche un treno che alla chiamata prelevi un cagnolino e al ritorno rilasci un gatto – vi sembrerà questo forse uno stile piuttosto fantasioso ma questo linguaggio (il C/C++) non si crea mica tanti scrupoli e a qualsiasi livello! - : le cose fortunatamente non stanno esattamente in questi termini, ma vi siamo molto vicini.

Oltre a ciò è necessario precisare che **ByVal** indica che noi desideriamo passare l’argomento per valore, mentre **ByRef** per riferimento, tutto come il VB normale: in fondo si tratta solo di funzioni!

Inoltre se vi trovati di fronte funzioni il cui nome terminano con **…ex**, nel caso in cui il SDK ve lo chieda espressamente, usate queste. Queste sono funzioni che rappresentano un aggiornamento delle corrispondenti senza il suffisso …ex in Win32, a volte con funzionalità in più.

Come chiamare le funzioni

Nel microesempio precedente abbiamo chiamato la funzione secondo la consuetudine, d’altra parte se proprio il valore di ritorno non ci fosse utile, avremmo potuto chiamarlo anche con **Call**, infatti la routine di conversione da stringa con zero a stringa pura da me proposta prevede già un controllo sulla fine del vettore, così potremo modificare il programma in questo modo con i medesimi risultati:

```
Dim Risult&
 …
Call GetWindowsDirectory (WinDir, 145)
Print "Directory Windows = " & StrNulToStr(WinDir, 0)
```

È più semplice, vero? Tenete presente a questo punto che molte volte il SDK vi chiederà proprio di tener in conto che

il risultato del vostra richiesta è prelevabile da una altra funzione: infatti capita spesso che i possibili errori vengano prelevati tramite la funzione **GetLastError(void)**.

I tipi

Sicuramente il maggior dubbio vostro è dato dal termine **lpbuffer**: *che sarà mai stà roba?* Questo è un **buffer** che ha come suffisso **lp**, cioè un puntatore a una struttura o voce di dati. Insomma questa è la notazione Ungherese. Nella tabella seguente vi sono i tipi e il prefisso corrispondente per Win32:

tipo	prefisso	commento	bit	tipo	prefisso	commento	bit
bool	b	boolean(= 0/1)	32	byte	ch	int unsigned	8
char	ch	int signed	8	tchar	ch	int signed	8/16
farproc	lpfn	puntatore far	32	handle	h	handle windows	32
int	n	int signed	32	short	n	int signed	16
long	l	int signed	32	lp	lp	pointer long	32
lpint	pli	pointer long	32	lpstr	lpsz	pointer long(nul str)	32
lpcstr	lpsz	pointer long (nul str)	32	lptstr	lpsz	pointer long (nul str)	32
np	n	non utilizzabile		npstr	np	non utilizzabile	
uint	n	int unsigned	32	word	w	int unsigned	16
dword	dw	int unsigned	32	flags	f	bit flag	16/32
lpunknow		puntatore	32				

Spiegare adesso in questo breve articolo tutto su questi tipi sarebbe un po' troppo, ma è già sufficiente sapere che sono così e che bisogna dare un'occhiata al C/C++ per comprendere meglio quanto detto. Risulta interessante legare questo concetto con quello definito nel modo VB, più esattamente quello della dichiarazione degli argomenti, in ciò risulta altamente utile il C/C++, poiché se ad esempio il tipo richiesto dalle API dovesse essere una UNSIGNED WORD (da 0 a 65535) a 16 bit fate bene attenzione che non ne esiste alcuno che corrisponda esattamente al VB. Perciò bisognerà applicare il tipo più vicino INTEGER (da -32768 a 32767) a 16 bit, di seguito una routine che svolga il compito di conversione che preveda la sottrazione di 32768 in qualsiasi caso, con una *operazione And* sui bit: ma questo non è un caso isolato! Il motivo sta nel fatto che se vi è stato chiesto proprio quel valore in bit, voi dovrete fornigliene uno identico a parità di bit, altrimenti vi trovereste con un valore non desiderato.

Per chi programma col C questo è uno dei problemi all'ordine del giorno, dato che è diverso confrontarsi con un computer ALFA o PC, il primo è a 64 bit su cui possono girare SO da 64 bit in giù, il secondo può essere a 32 o 16 bit con lo stesso effetto: che cosa succederebbe se il vostro programma volesse accedere all'hardware bypassando il SO e voi vorreste farlo eseguire a tutt'e due? Tenete presente che ambedue possono supportare Windows NT. Perciò è vostro compito sempre dare un'occhiata ai valori dei tipi in VB espressi in bit sul vostro manuale.

Un discorso a parte invece meritano **i tipi plurimi as Any**, verso i quali si sarebbe tentati di utilizzare i tipi Variant, dato che traducendo letteralmente dall'inglese per noi potrebbe significare *tipo qualsiasi*, fidandosi in tal modo dell'estrema leggerezza con cui il C tratta i dati e del fatto che, talvolta essendoci andata bene, ciò ci ha fatto credere che sia proprio il VB a occuparsi di tutto!

In parte l'affermazione è vera, poiché il VB di fatto ha un buon controllo su questi tipi di chiamate, adattandosi ai nostri desideri: ma non è sempre così. Prendete ad esempio questa funzione: Declare Function LoadCursor Lib "user32" Alias "LoadCursorA" (ByVal hInstance As Long, ByVal lpCursorName As Any) As Long, essa accetta sia una *string* che una long, con due servizi totalmente differenti: diamine, se qui vi sbagliate ci lascerete veramente le penne!

Dunque che cosa sono gli argomenti?

Gli argomenti servono per l'**immissione**, il **prelievo** o l'**impostazione di dati**, non necessariamente in questa sequenza e neppure tutte insieme: dipende solo dal servizio e dal tipo di progettazione pensato dalla Microsoft. Questo lo si comprenderà meglio con l'esempio successivo, per ora bisogna tenere presente che solo uno sparuto gruppo di funzioni di fatto restituisce qualcosa immediatamente, la maggior parte svolge un oscuro ed umile compito che ha senso solo nel gruppo di appartenenza: grafica, contesti di dispositivo, menu, file, processi, etc.

Impariamo ad usare le strutture e le costanti

Come sopra, tramite il Visualizzatore testo API, inseriamo nel nostro modulo standard la funzione GetSystemInfo(…). Subito si potrà notare contrariamente a quella precedente che l'unico argomento presente è scritto tutto in maiuscolo: questa è una convezione utilizzata per avvertirvi che state utilizzando una struttura, cioè una **struct** in C/C++, una **type** in Delphi o VB. Adesso dovrete solo richiamarle ambedue col Visualizzatore testo API, la prima nella ComboBox Tipo API tramite la voce Dichiarazioni, la seconda tramite quella dei tipi, di seguito inserirle nel vostro modulo standard. Questa struttura prevede anche alcune costanti che si riferiscono al tipo di processore presente, esse sono dei codici identificativi che le API ci danno che bisogna confrontare per conoscere il tipo della CPU; contrariamente a quanto detto in

precedenza per gli altri casi, sebbene la procedura di inserimento nel nostro modulo standard avvenga sempre col *Visualizzatore Testo API*, selezionando nella ComboBox *Tipo API* la voce **Costanti**, adesso noi conosciamo la loro esistenza e il loro uso solo leggendo quanto il SDK ci dice rispetto alla struttura. Per il resto bisogna assegnare una variabile public del tipo della nostra struttura (non potete usare direttamente essa nei vostri programmi), di seguito chiamare con la stessa la funzione e stampare i risultati, confrontando le costanti col campo riferito al processore. Eccole qui:

```
Type SYSTEM_INFO
        dwOemID As Long                         'restituisce il processore usato: 0 in Win95
        dwPageSize As Long                          ' restituisce la grandezza della paginazione usata
        lpMinimumApplicationAddress As Long         'Indirizzo più basso per un'applicazione
        lpMaximumApplicationAddress As Long         'Indirizzo più alto per un'applicazione
        dwActiveProcessorMask As Long           'numero di processori configurati nel sistema
        dwNumberOrfProcessors As Long           'numero di processori nel sistema
        dwProcessorType As Long                 'Tipo di Processore, obsoleta, ma adesso utile
        dwAllocationGranularity As Long         'composizione memoria virtuale: 64k default
        dwReserved As Long                          'riservato
End Type

Declare Sub GetSystemInfo Lib "kernel32" Alias "GetSystemInfo" (lpSystemInfo As SYSTEM_INFO)
```

Il terzo esempio incluso nel Cd-Rom vi mostra l'uso di tale funzione, di essa non dovete far altro che richiamarla per ottenere le informazioni desiderate.

Una dichiarazione con più argomenti

Con la funzione GetVolumeInformation(…), che restituisce le caratteristiche della memoria di massa richiesta, si vedrà praticamente l'uso di argomenti compositi, evitando adesso di ripeterci su cose già dette su. Accanto ad ogni argomento v'è la spiegazione, che dovrebbe essere già sufficiente per comprensione immediata di essa.

```
Declare Function GetVolumeInformation Lib   "kernel32" Alias "GetVolumeInformationA"
  (ByVal lpRootPathName As String,          'inserire: unità espressa in lettere "A:" , "B:", "C", "D:" ….."
  ByVal lpVolumeNameBuffer As String,       'prelevare: l'etichetta dell'unità (stringa terminante con zero)
  ByVal nVolumeNameSize As Long,            'inserire: 255 grandezza del vettore in lpVolumeNameBuffer
  lpVolumeSerialNumber As Long,             'prelevare: numero di serie dell'unità
  lpMaximumComponentLength As Long,         'prelevare: grandezza massima di un nome di file predefinita dal SO
  lpFileSystemFlags As Long,                'prelevare: tipo di File System: compresso, unicode,…
                                            'da reperire tramite costanti
  ByVal lpFileSystemNameBuffer As String,   'prelevare: nome del FileSystem
  ByVal nFileSystemNameSize As Long)        'inserire: 255 grandezza del vettore in lpFileSystemNameBuffer
As Long                                     '    riuscita dell'operazione: 1 = OK, 0 = fallita
```

Come al solito nel CD-Rom allegato c'è l'esempio quarto al riguardo

Considerazioni finali

Probabilmente si potrà pensare di unire tutte le chiamate API in un unico modulo per utilizzarle facilmente nei propri programmi. Questa non è un'idea malsana che tuttavia non tiene conto del fatto che esse per poter funzionare adeguatamente hanno comunque bisogno delle Dichiarazioni dei Tipi e delle Costanti, cosa che in una **memoria Heap** (cioè quella parte della memoria riservata ad ogni applicazione ai dati, determinata dal SO tramite le informazioni contenute nell'**Header** [**intestazione**] del file) predefinita del VB non riuscirebbero a collocarsi e il programma nemmeno partirebbe.

Se avete pensato a costruirci su una DLL da VB, tenete conto del fatto che essa non è affatto identica a quelle che voi avete chiamato tramite l'istruzione DECLARE, poiché il VB le crea con un sistema pressappoco simile agli Overlay (OVR) usati in alcuni linguaggi nel DOS, con l'evidente conseguenza di condividere la stessa memoria Heap usata dal vostro programma. Allora perché utilizzare le API da VB?

Di fatto il VB offre numerose estensioni al proprio linguaggio, le quali coprono buona parte delle normali richieste dei programmatori. Le stesse vi vengono offerte direttamente senza bisogno di dover accedere a tool quali le MFC o gli Object Windows, che il più delle volte sono facili da implementare, ma complicati da gestire, poiché per esse, se si desidera apportare modifiche personali, bisogna sempre risalire alle **classi parent** per conoscerne bene la struttura e poi implementare le nostre istanze o funzioni membro ereditando le classi stesse: il VB fortunatamente ci mette al riparo da tutto ciò, rendendoci tutto più semplice.

Le API possono velocizzare molto in alcuni casi il alcune operazioni del nostro programma o darci delle informazioni che non possono essere reperite senza esse stesse. Prendete il caso successivo e ve ne renderete conto.

Dovevo caricare un database di 5000 elementi circa in una ComboBox. Per evitare che il tempo necessario per

compiere tale operazione facesse credere all'utente un possibile errore del programma, bisognava implementare una barra di avanzamento grafica e attivare la funzione DoEvents() in modo da liberare il SO dall'eccessiva presenza del programma in esecuzione. A questo punto l'unica cosa da fare era velocizzare questa procedura tramite funzioni API che caricassero gli elementi del database velocemente nella ComboBox e costruire velocemente la barra di avanzamento, tramite le funzioni grafiche, evitando l'uso (in questo caso lento) del controllo line.

Probabilmente penserete a questo punto che vale la pena di utilizzare un altro linguaggio: questo potrebbe essere vero rispetto all'uso specifico che se ne deve fare e alle capacità offerte dal linguaggio stesso. Ma se le vostre conoscenze di programmazione non volete che diventino il fine della vostra vita, bisogna scegliere un linguaggio semplice.

D'altra parte, se il tutto il resto del vostro programma l'avete creato in cinque minuti col VB, che senso ha pensare ad un altro linguaggio per crearvi inutili problemi?

Resta comunque il fatto che con le API bisogna avere pazienza e legger bene il SDK prima di implementarle.

Nel CD-Rom allegato alla rivista n° 5 di ioProgrammo c'è un file, che ritengo troverete molto utile*: msvbkb.exe*. Mentre stavo scrivendo quest'articolo nella rivista n° 14 ho trovato il *Microsoft platform SDK* con relativo articolo di presentazione: questa è una vera chicca, poiché tutto ciò che potrete creare con le API si trova proprio in essi (SDK) e in più potrete operare il porting delle vostre applicazioni per Win98/WinNT5.

Vi ringrazio di avermi seguito e spero solo di aver proposto l'argomento con semplicità.

A presto. Prof. Francesco Mannarino