Trucos y rutinas para

Visual Basic

Contenido:

Otra forma de usar VScroll y HScroll...

Notas

Notas:

Todos estos ejemplos y rutinas son de libre uso.

Si tienes algunos que quieras que se añadan, sólo tienes que enviarmelo por e−mail

Cuando haya una cantidad más o menos "considerable", veré de crear un fichero de ayuda.

Cualquier comentario SIEMPRE es bienvenido.

Gracias por colaborar.

1.−Mover un Form sin caption ¡Al fin un método sencillo!

```
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
'NOTAS:
'Listado a insertar en un módulo (.bas)
'si se quiere poner en un formulario (.frm)
'declarar la función como Private y quitar el Global de las constantes
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
'Constantes y declaración de función:
'
'Constantes para SendMessage
```

```
Global Const WM_LBUTTONUP = &H202

Global Const WM_SYSCOMMAND = &H112

Global Const SC_MOVE = &HF010

Global Const MOUSE_MOVE = &HF012

#If Win32 Then

Declare Function SendMessage Lib "User32" Alias "SendMessageA" (ByVal hwnd As Long, ByVal wMsg
As Long, ByVal wParam As Long, lParam As Long) As Long

#Else

Declare Function SendMessage Lib "User" (ByVal hWnd As Integer, ByVal wMsg As Integer, ByVal
wParam As Integer, lParam As Any) As Long

#End If

'

'

'Este código se pondrá en el Control_MouseDown...

'

Dim lngRet As Long

'Simular que se mueve la ventana, pulsando en el Control

If Button = 1 Then

'Envía un MouseUp al Control

lngRet = SendMessage(Control.hWnd, _

WM_LBUTTONUP, 0, 0)

'Envía la orden de mover el form

lngRet = SendMessage(FormX.hWnd, _

WM_SYSCOMMAND, MOUSE_MOVE, 0)

End If
```

2.−Mover y soltar controles con Drag & Drop (AL FIN!)

```
'––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

'Me ha costado cogerle el tranquillo al tema del Drag & Drop,

'ya que los ejemplos no ayudaban mucho para lo que yo lo quería.

'Se usan: DragOver, DragDrop, MouseDown y MouseUp.

'El único coñazo es tener que poner código en todos los controles...

'––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

'Variables a nivel del módulo

Dim DY As Single

Dim DX As Single

Private Sub CancelarDrag(Source As Control)

Source.Visible = True

Source.Drag vbCancel

End Sub

Private Sub FinalizarDrag(Source As Control, Button As Integer)

If Button = vbLeftButton Then

Source.Visible = True

Source.ZOrder

Source.Drag vbEndDrag

End If

End Sub

Private Sub IniciarDrag(Source As Control, Button As Integer, X As Single, Y As Single)

If Button = vbLeftButton Then

DX = X

DY = Y

'Permitir la operación de Drag & Drop

Source.Drag vbBeginDrag
```

```
'Cambiar a no visible, ya que si no, el form no detectaría que se ha soltado, si el puntero del ratón no sale del control.

Source.Visible = False

'Comienza el espectáculo

Source.Drag

End If

End Sub

Private Sub Form_DragDrop(Source As Control, X As Single, Y As Single)

'Si se quieren excluir algunos controles,

'hacer aquí la comparación.

Source.Visible = True

Source.Move X − DX −60, Y − DY −60

Source.Drag vbEndDrag

Source.ZOrder

End Sub

'En cada control poner este código:

(cambiar %Control% por el nombre apropiado)

'

Private Sub %Control%_DragDrop(Source As Control, X As Single, Y As Single)

CancelarDrag Source

End Sub

'

Private Sub %Control%_MouseDown(Button As Integer, Shift As Integer, X As Single, Y As Single)

IniciarDrag %Control%, Button, X, Y

End Sub

'

Private Sub %Control%_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
```

```
FinalizarDrag %Control%, Button

End Sub

'

'Se puede añadir DragOver para que muestre un icono no permitiendo que se suelte.

'
```

3.−Cambiar el tamaño de un Picture usando el API de Windows

```
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'Redimensionar un Picture usando el API de Windows

'Funciones usadas: GetWindowLong, SetWindowLong y SetWindowPos

'El ejemplo tiene en el Form los siguientes objetos:

'Label1() y Text1() en cada PicColumn()

'Label2() en el form

'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'

'

Option Explicit

'Prueba para redimensionar Pictures

Dim NumColumnas As Integer

Dim NumFilas As Integer

Dim bIniciando As Boolean

#If Win32 Then

Private Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" (ByVal hwnd As Long,
ByVal nIndex As Long) As Long

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

Private Declare Function SetWindowPos Lib "user32" (ByVal hwnd As Long, ByVal hWndInsertAfter As
```

```
Long, ByVal X As Long, ByVal Y As Long, ByVal cX As Long, ByVal cY As Long, ByVal wFlags As
Long) As Long

#Else

Private Declare Function GetWindowLong Lib "User" (ByVal hwnd As Integer, ByVal nIndex As Integer) As
Long

Private Declare Function SetWindowLong Lib "User" (ByVal hwnd As Integer, ByVal nIndex As Integer,
ByVal dwNewLong As Long) As Long

Private Declare Function SetWindowPos Lib "User" (ByVal hwnd%, ByVal hWndInsertAfter%, ByVal X%,
ByVal Y%, ByVal cX%, ByVal cY%, ByVal wFlags%) As Integer

#End If

Const GWL_STYLE = (−16)

Const WS_THICKFRAME = &H40000

Const WS_CHILD = &H40000000

Const SWP_DRAWFRAME = &H20

Const SWP_NOMOVE = &H2

Const SWP_NOSIZE = &H1

Const SWP_NOZORDER = &H4

Private Sub Form_Load()

Dim Style as Long

bIniciando = True

Style = GetWindowLong(PicColum(0).hwnd, GWL_STYLE)

Style = Style& Or WS_THICKFRAME

Style = SetWindowLong(PicColum(0).hwnd, GWL_STYLE, Style)

Style = SetWindowPos(PicColum(0).hwnd, _

Me.hwnd, 0, 0, 0, 0, SWP_NOZORDER Or _

SWP_NOSIZE Or SWP_NOMOVE Or SWP_DRAWFRAME)

NumFilas = 2

Load Text1(1)
```

```
Set Text1(1).Container = PicColum(0)

Text1(1).Visible = True

Text1(1).Top = Text1(0).Top + Text1(0).Height

Load Label2(1)

Label2(1).Visible = True

Label2(1).Top = Label2(0).Top + Label2(0).Height

Label2(1) = "Fila 2"

NumColumnas = 1

bIniciando = False

End Sub

Private Sub PicColum_Resize(Index As Integer)

Dim k As Integer

Dim i As Integer

If bIniciando Then Exit Sub

'ajustar el ancho del Label y los texts

Label1(Index).Width = PicColum(Index).Width

For i = 0 To NumFilas − 1

k = i * NumColumnas + Index

Text1(k).Width = PicColum(Index).Width

Next

PicColum(0).Left = Label2(0).Width

For i = 0 To NumColumnas − 1

If i > 0 Then

PicColum(i).Left = PicColum(i − 1).Left + PicColum(i − 1).Width

End If

PicColum(i).Top = 0
```

```
Next

End Sub
```

4.−Métodos para usar el CommonDialog de Visual Basic

```
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'Ejemplos de los métodos para Seleccionar Impresora, Abrir, Guardar

'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'Seleccionar impresora

On Local Error Resume Next

CommonDialog1.CancelError = True

CommonDialog1.Flags = cdlPDPrintSetup

CommonDialog1.ShowPrinter

Err = 0

'Abrir

On Local Error Resume Next

CommonDialog1.CancelError = True

'Especificar las extensiones a usar

CommonDialog1.DefaultExt = "*.crd"

CommonDialog1.Filter = "Cardfile (*.crd)|*.crd|Textos (*.txt)|*.txt|Todos los archivos (*.*)|*.*"

CommonDialog1.ShowOpen

If Err Then

'Cancelada la operación de abrir

Else

sArchivo = CommonDialog1.FileName

End If

'Guardar
```

```
On Local Error Resume Next

CommonDialog1.CancelError = True

'Especificar las extensiones a usar

CommonDialog1.DefaultExt = "*.crd"

CommonDialog1.Filter = "Cardfile (*.crd)|*.crd|Textos (*.txt)|*.txt|Todos los archivos (*.*)|*.*"

CommonDialog1.FileName = sArchivo

CommonDialog1.ShowSave

If Err Then

'Cancelada la operación de guardar

Else

sArchivo = CommonDialog1.FileName

End If
```

5.−Crear controles que se pueden cambiar de tamaño usando el API de Windows

```
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'Convertir controles en VENTANAS. Poder cambiar el tamaño, etc.

'Funciones usadas: GetWindowLong, SetWindowLong y SetWindowPos

'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'

'Declaraciones globales a nivel de módulo

'

#If Win32 Then

Declare Function GetWindowLong Lib "user32" Alias "GetWindowLongA" (ByVal hWnd As Long, ByVal
nIndex As Long) As Long

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

Declare Function SetWindowPos Lib "user32" (ByVal hWnd As Long, ByVal hWndInsertAfter As Long,
```

```
ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As
Long

#Else

Declare Function GetWindowLong Lib "User" (ByVal hwnd As Integer, ByVal nIndex As Integer) As Long

Declare Function SetWindowLong Lib "User" (ByVal hwnd As Integer, ByVal nIndex As Integer, ByVal
dwNewLong As Long) As Long

Declare Function SetWindowPos Lib "User" (ByVal hwnd%, ByVal hWndInsertAfter%, ByVal X%, ByVal
Y%, ByVal cX%, ByVal cY%, ByVal wFlags%) As Integer

#End If

Global Const GWL_STYLE = (−16)

Global Const WS_THICKFRAME = &H40000

Global Const WS_CHILD = &H40000000

Global Const SWP_DRAWFRAME = &H20

Global Const SWP_NOMOVE = &H2

Global Const SWP_NOSIZE = &H1

Global Const SWP_NOZORDER = &H4

Private Sub Form_Load()

Dim Style&, ret&

'Cambiar %Control% por el control a usar: (Text, Picture...)

Style& = GetWindowLong(%Control%.hWnd, GWL_STYLE)

Style& = Style& Or WS_THICKFRAME

Style& = SetWindowLong(%Control%.hWnd, GWL_STYLE, Style&)

ret& = SetWindowPos(%Control%.hWnd, _

Me.hWnd, 0, 0, 0, 0, SWP_NOZORDER Or _

SWP_NOSIZE Or SWP_NOMOVE Or SWP_DRAWFRAME)

End Sub
```

6.−Extraer iconos usando librerías del API de Windows

```
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'Extraer iconos de una aplicación o librería y dibujarlo en un picture.

'Usando librerías del Api de Windows (ExtractIcon GetClassWord DrawIcon)

'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'

'Declaraciones para extraer iconos de los programas

'

'Versión 32 bits

'

'hIcon el número de icono a extraer, el 0 es el primero.

Declare Function ExtractIcon Lib "shell32.dll" Alias "ExtractIconA" (ByVal hInst As Long, ByVal
lpszExeFileName As String, ByVal nIconIndex As Long) As Long

Declare Function GetClassWord Lib "user32" Alias "GetClassWord" (ByVal hwnd As Long, ByVal nIndex
As Long) As Long

Declare Function DrawIcon Lib "user32" Alias "DrawIcon" (ByVal hdc As Long, ByVal x As Long, ByVal y
As Long, ByVal hIcon As Long) As Long

Const GCW_HMODULE = (−16&)

Function ExtraerIcono (quePicture As Integer, sPrograma As String, queIcon As Long) As Long

'Cargar el icono del programa

Dim myhInst As Long

Dim hIcon As Long

Dim i As Long

myhInst = GetClassWord(hWnd, GCW_HMODULE)

hIcon = ExtractIcon(myhInst, sPrograma, queIcon)

If hIcon Then

Picture1(quePicture).Picture = LoadPicture("")

Picture1(quePicture).AutoRedraw = −1
```

```
i = DrawIcon(Picture1(quePicture).hDC, 0, 0, hIcon)

Picture1(quePicture).Refresh

End If

ExtraerIcono = hIcon

End Function

'

'Versión para 16 bits

'

'hIcon el número de icono a extraer, el 0 es el primero.

Declare Function ExtractIcon Lib "Shell" (ByVal hInstance As Integer, ByVal pszExeName As String, ByVal hIcon As Integer) As Integer

Declare Function GetClassWord Lib "User" (ByVal hWnd As Integer, ByVal nIndex As Integer) As Integer

Declare Function DrawIcon Lib "User" (ByVal hDC As Integer, ByVal x As Integer, ByVal Y As Integer, ByVal hIcon As Integer) As Integer

Const GCW_HMODULE = (−16)

Function ExtraerIcono (quePicture As Integer, sPrograma As String, queIcon As Integer) As Integer

'Cargar el icono del programa

Dim myhInst As Integer

Dim hIcon As Integer

Dim i As Integer

myhInst = GetClassWord(hWnd, GCW_HMODULE)

hIcon = ExtractIcon(myhInst, sPrograma, queIcon)

If hIcon Then

Picture1(quePicture).Picture = LoadPicture("")

Picture1(quePicture).AutoRedraw = −1

i = DrawIcon(Picture1(quePicture).hDC, 0, 0, hIcon)

Picture1(quePicture).Refresh
```

```
End If

ExtraerIcono = hIcon

End Function
```

7.−Añadir a la lista de un Combo el texto escrito

```
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'Añadir a la lista de un combo, el texto escrito, si es que no está.

'Usarlo del tipo: 0−DropDown Combo

'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

Sub ActualizarCombo()

'Actualizar el contenido del Combo

Dim sTmp As String

Dim i As Integer

Dim j As Integer

Dim hallado As Boolean

Dim k As Integer

For k = 0 To 1

hallado = False

sTmp = Combo1(k).Text

If Len(Trim$(sTmp)) Then

j = Combo1(k).ListCount − 1

For i = 0 To j

If StrComp(Trim$(sTmp), Trim$(Combo1(k).List(i))) = 0 Then

hallado = True

Exit For

End If
```

```
Next
If Not hallado Then
Combo1(k).AddItem sTmp
End If
End If
Next
End Sub
```

8.−Imitar un Combo Box al estilo del de ayuda.

```
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
'Para imitar un ComboBox parecido al de Buscar en Ayuda de Windows,
'(va cambiando según las letras escritas).
'El form debe tener un Textbox y un Listbox.
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
'
'Código en un Módulo (.BAS):
Option Explicit
Global CHClickList As Integer
Global CHInChange As Integer
Sub CtrlTB_Change (OTB As TextBox, OLB As ListBox)
Dim Pos As Integer, I As Integer, L As Integer
Dim Aux As String
If CHClickList Then
CHClickList = False
Exit Sub
End If
```

```
Aux = OTB.Text

L = Len(Aux)

For I = 0 To (OLB.ListCount − 2)

If Not StrComp(Aux, Left$(OLB.List(I), L), 1) > 0 Then

Exit For

End If

Next I

OLB.TopIndex = I

OLB.ListIndex = I

End Sub

Sub CtrlTB_KeyPress (OTB As TextBox, OLB As ListBox, KeyAscii As Integer)

If KeyAscii = 13 Then

OTB.Text = Left$(OLB.List(OLB.ListIndex), 60)

CHInChange = False

Else

CHInChange = True

End If

End Sub

Sub CtrlLB_Click (OTB As TextBox, OLB As ListBox)

If Not CHInChange Then

OTB.Text = Left$(OLB.List(OLB.ListIndex), 60)

Else

CHInChange = False

End If

End Sub

Sub CtrlLB_MouseDown ()
```

```
CHClickList = True

End Sub

'Código en el Form (.FRM):

Sub List1_Click ()

CtrlLB_Click Text1, List1

End Sub

Sub List1_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)

CtrlLB_MouseDown

End Sub

Sub Text1_Change ()

CtrlTB_Change Text1, List1

End Sub

Sub Text1_KeyPress (KeyAscii As Integer)

CtrlTB_KeyPress Text1, List1, KeyAscii

End Sub
```

9.−Scroll horizontal para un List Box usando SendMessage

```
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'Como poner una barra de scroll horizontal en un List Box.

'"Truco" tomado de Microsoft Knowledge Base Articles.

'How to Add a Horizontal Scroll Bar to Visual Basic List Box; Article ID: Q80190

'Función: SendMessage

'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'Declaraciones de las funciones para 16 y 32 bits

'Para 16 bits (VB3 y VB4)

Declare Function SendMessage Lib "user" (ByVal hWnd%, ByVal wMsg%, ByVal wParam%, ByVal
```

```
lParam&) As Integer

'

'Para 32 bits usar:

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

'

'

'Poner en Form_Activate

Const LB_SETHORIZONTALEXTENT = &H400 + 21

Const NULO = &O0

Dim ListhWnd As Integer 'Handle del List Box

Dim ListLen As Integer 'Ancho del List Box

Dim iTmp As Integer 'Para el valor devuelto por SendMessage

Dim ScaleTmp As Integer 'Valor anterior de ScaleMode

ScaleTmp = ScaleMode

ScaleMode = 3 'wParam is in PIXEL(3)

ListhWnd = List1.hWnd

ListLen = 32767 'TextWidth(String$(256, "A"))

iTmp = SendMessage(ListhWnd, LB_SETHORIZONTALEXTENT, ListLen, NULO)

ScaleMode = ScaleTmp 'Restablecer el valor anterior de ScaleMode
```

10.−TextBox con 64 KB en lugar de 32 KB

```
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'Usando SendMessage del Api de Windows, poder tener text−box con 64 KB

'en lugar de los 32 que admite Visual Basic.

'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```

'Declaración de la función API

Declare Function sendMessage Lib "User" (ByVal hWnd As Integer, ByVal wMsg As Integer, ByVal wParam As Integer, lParam As Any) As Long

'

'Para 32 bits usar:

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

'

'Declaración de las constantes

Global Const WM_USER = &H400

Global Const EM_LIMITTEXT = WM_USER + 21

'En el Form_Load del text−box:

Dim LTmp as long

LTmp=SendMessage(Text1.hWnd,EM_LIMITTEXT,0,byval 0&)

11.−Comprobar si un programa cargado con Shell está ejecutandose

'−−−

'Por ser extenso para un "simple" truco, los ejemplos están comprimidos

'También se muestra como asignar el icono de un programa a un picture

'Hay un fichero para VB4 (16 y 32 bits) y otro para VB3

'−−−

La idea básica es:

1.− Cargar el programa usando Shell

2.− Comprobar si aún está activo (bucle)

3.− Continuar el programa principal una vez finalizado el programa cargado con Shell

Las funciones del API de Windows utilizadas son:

Para extraer el icono del programa:

ExtractIcon

GetClassWord

DrawIcon

Para comprobar las ventanas activas:

GetWindow

GetWindowText

GetWindowTextLength

IsWindowVisible

Baja los ejemplos del truco 11: Shell_t.zip (11.606 bytes)

12.− Catálogo de CD's musicales

Ejemplo para leer el volumen de un disco, esta función se puede usar para ¡catalogar los CD's musicales!

```
Declare Function GetVolumeInformation Lib "Kernel32" Alias "GetVolumeInformationA" (ByVal
lpRootPathName As String, ByVal lpVolumeNameBuffer As String, ByVal nVolumeNameSize As Long,
lpVolumeSerialNumber As Long, lpMaximumComponentLength As Long, lpFileSystemFlags As Long,
ByVal lpFileSystemNameBuffer As String, ByVal nFileSystemNameSize As Long) As Long

Dim lVSN As Long, n As Long, s1 As String, s2 As String

s1=String$(255,Chr$(0))

s2=String$(255,Chr$(0))

l= GetVolumeInformation("unidad", s1, Len(s1), lVSN, 0, 0, s2, Len(s2))

'lVSN tendrá el valor del Volume Serial Number (número de serie del volumen)
```

Si "unidad" es el CD−ROM y tenemos un disco de música, podemos usar el VSN para hacer un catálogo de CD's ya que cada CD tiene un número diferente.

Para comprobar si es un CD−ROM (o CD−musical):

```
' Valores de retorno de GetDriveType

Public Const DRIVE_REMOVABLE = 2

Public Const DRIVE_FIXED = 3
```

```
Public Const DRIVE_REMOTE = 4

Public Const DRIVE_CDROM = 5

Public Const DRIVE_RAMDISK = 6

Declare Function GetDriveType Lib "Kernel32" Alias "GetDriveTypeA" (ByVal nDrive As String) As Long

Dim lDrive As Long

Dim szRoot As String

szRoot="D:\" 'Poner aquí la unidad del CD−ROM o la que queramos comprobar

lDrive= GetDriveType(szRoot)

If lDrive = DRIVE_CDROM Then

'Es un CD−ROM/Compact−Disc

End If
```

15.− Seleccionar el texto al entrar en un TextBox

Este truco, creo que es conocido por todos, pero lo "recuerdo" por si hay alguno no lo sabe...

```
'Para un control

Private Sub Text1_GotFocus()

Text1.SelStart = 0

Text1.SelLength = Len(Text1)

End Sub

'Para un array

Private Sub Text1_GotFocus(Index As Integer)

Text1(Index).SelStart = 0

Text1(Index).SelLength = Len(Text1(Index))

End Sub
```

16.− Mostrar la posición del cursor en un TextBox

Este truco, muestra la posición actual del cursor y la longitud total del TextBox. Por supuesto el tamaño máximo permitido, debemos asignarlo a Text1.MaxLength, yo lo uso en mis programas, para saber cuando tengo que empezar a abreviar lo que estoy escribiendo, no siempre se dispone de todo el espacio que uno quiere, sobre todo cuando no quieres que las bases de datos se hagan enormes!

'Se puede cambiar StatusBar por cualquier control que nos muestre la información...

Private Sub Text1_Click()

miForm!StatusBar1.Panels("Posic").Text = " Pos: " & Text1.SelStart + 1 _

& "/" & Text1.MaxLength

End Sub

Private Sub Text1_KeyUp(KeyCode As Integer, Shift As Integer)

miForm!StatusBar1.Panels("Posic").Text = " Pos: " & Text1.SelStart + 1 _

& "/" & Text1.MaxLength

End Sub

17.− Refrescar el contenido de un control con DoEvents

¿Cuantas veces has asignado a un Label un nuevo Caption y no lo ha mostrado?, prueba a poner DoEvents después de la asignación y verás como se muestra enseguida.

Puedes usar *Sleep 0&* en lugar de DoEvents. La explicación de este consejo.

18.− Mostrar el texto de un CheckBox seleccionado cuando está marcado

Bueno, esto no es realmente un truco, pero podría serlo.

Cuando seleccionamos una opción de un CheckBox, algunas veces, nos puede interesar que el texto se quede "marcado".

Por ejemplo, si quisieramos hacer un list box al estilo del que viene con las FM 2.0 de Microsoft. Y que seguramente estará (o ya está?) en VB5

El truco consiste en cambiar el color del checkbox cuando este está seleccionado.

Private Sub Check1_Click()

```
If Check1 Then

Check1.ForeColor = colForeSelect

Check1.BackColor = colBackSelect

Else

Check1.ForeColor = colForeNormal

Check1.BackColor = colBackNormal

End If

End Sub
```

Las variables colForeSelect, colBackSelect, colForeNormal, colBackNormal, deben estar definidas con los colores que queramos usar. Por ejemplo:

```
Dim colBackNormal As Long

Dim colForeNormal As Long

Dim colBackSelect As Long

Dim colForeSelect As Long

colBackNormal = Check1.BackColor

colForeNormal = QBColor(0) 'Negro

colBackSelect = QBColor(1) 'Azul

colForeSelect = QBColor(15) 'Blanco brillante
```

Ejemplo de chk extendido (eje_chk1.zip 1.883 bytes)

19.− Crear una lista de CheckBox, ChkList

Este tipo de control existe en VB5 pero no en los anteriores, salvo que sea en un VBX/OCX externo.

De lo que se trata es de simular un ListBox, pero en lugar de usar sólo un texto como contenido, se usa un CheckBox. En los listados que se acompañan, hay también un ejemplo de cómo crear un panel deslizable (Picture con Scroll). Para que al mover el scroll vertical u horizontal, se desplace el contenido del CheckList, realmente esta es "la madre del cordero". También he creado un Picture dimensionable, usando el API de Windows, para poder cambiar "manualmente" el tamaño del contenedor del ChkList en tiempo de ejecución.

Listado del ejemplo de ChkList (chklist.zip 2.811 bytes)

20.− Usa tu computadora para ganar dinero fácil y rápido...

De nuevo Joe LeVasseur... La rutina es para saber si puedes ganar dinero rápido... sin hacer nada.

```
Public Function Dinero_Rapido() As Boolean

Dim Tonto

Dim No_Quiere_Trabajar

If No_Quiere_Trabajar And Tonto Then

Dinero_Rapido = True

Else

Dinero_Rapido = False

Tonto = False

End If

End Function

Private Sub Command1_Click()

Print Dinero_Rapido

End Sub

'Pruebalo, siempre tiene el mismo resultado.
```

Bueno, como comprenderás, se trata de una broma. Esta "rutina" fue la respuesta de Joe a Jorge E. Mora en las news, a la propuesta de éste último para ganar **$$$$$ DINERO RAPIDO $$$$$$**

Te prometo que el próximo truco será de "verdad."

21.− Otra forma de usar VScroll y HScroll...

En realidad es comentar que si al asignar los valores Mínimos y Máximos de estos controles de manera que el valor Máximo sea inferior al Mínimo, se desplazarán al revés.

Cuando se usa de la forma habitual, al pulsar en la flecha superior del VScroll, el valor disminuye.

De esta otra forma, al pulsar arriba, se incrementa.

ir al

Contenido:

Notas:

Todos estos ejemplos y rutinas son de libre uso.

Si tienes algunos que quieras que se añadan, sólo tienes que enviarmelo por e−mail

Cuando haya una cantidad más o menos "considerable", veré de crear un fichero de ayuda.

Cualquier comentario SIEMPRE es bienvenido.

Gracias por colaborar.

1.− ¿Recursos?: Si, Gracias!

Pues el truco con el que empiezo este nuevo archivo es para simular un Frame usando Shape.

Con lo cual, el consumo de recursos del sistema, creo, será menor.

Usa el control Shape y dibuja 2 en el form. dale el tamaño y la posición que quieras, pero uno encima del otro. Al primero le pones BorderWidth=2 y el color negro. Al segundo lo dejas con BorderWidth=1, pero el color blanco. Debe estar el segundo encima del primero, para que haga el efecto 3D.

Fácil, verdad?

El único problema es que si incluyes controles en el interior, para moverlos, no es tan fácil cómo si usaras un frame, pero...

En el programa que incluyo hoy, hay ejemplo de esto que estoy diciendo.

2.− Comprobar cómo se cierra una aplicación

Al cerrar un form, podemos saber si es nuestro código el que cierra la aplicación o bien se cierra por otra causa.

Esta comprobación se hace en Form_QueryUnload y puede ser:

QueryUnload Method

Constant Value Description

vbFormCode 1 Unload method invoked from code.

vbAppWindows 2 Current Windows session ending.

vbFormMDIForm 4 MDI child form is closing because the MDI form is closing.

vbFormControlMenu 0 User has chosen Close command from the Control−menu box on a form.

vbAppTaskManager 3 Windows Task Manager is closing the application.

'Ejemplo para usarlas:

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)

'Sólo cerrar si es un mensaje de windows

Select Case UnloadMode

Case vbFormCode, vbAppTaskManager, vbAppWindows

'ok, cerrar

Case Else

MsgBox "No se permite cerrar la aplicación.", vbInformation, "Mensajes"

Cancel = True

WindowState = vbMinimized

End Select

End Sub

3.− Averiguar el signo decimal (coma o punto) (18/Feb)

Esto lo he usado para el programa de la calculadora y lo copié de un ejemplo que venía con el Visual Basic para MS−DOS

El listado, dejo hasta los comentarios en inglés, para que no digan que me quiero apuntar el tanto.

' Determine whether "." or "," should be used as

' decimal separator based on value returned by

' FORMAT$ (country specific).

```
temp$ = Format$(1.5, "#.#")

If InStr(temp$, ",") Then

Decimal = ","

Else

Decimal = "."

End If
```

4.− Usar los IO Ports en con VB 16 y 32 bits (26/Feb)

He "bajado" unas librerías de http://www.softcircuits.com/ con rutinas para manejar los puertos de entrada/salida, además de otras cosillas. Esto hay que agradecerselo, además de a la gente de softcircuits, a Victor Limiñana, ya que gracias a una consulta que me hizo sobre este tema, he podido encontrar estas librerías.

Además de los archivos comprimidos con, en algunos casos, ejemplos de cómo usarlos y hasta el código C para crear las librerías, me he tomado la libertad de poner, en el original inglés, los archivos LEEME que acompañan a dichas librerías. Espero que os sirva de algo.

La librería y ejemplos para 16 bits (vbhlp16.zip 37.962 bytes)

El contenido del archivo Vbhelper16.txt

La librería de varias utilidades para 32 bits y ejemplos (vbhlp32.zip 30.945)

El contenido del archivo Vbhlp32.txt

La librería para IO en Windows95, **no sirve para NT** (win95IO.zip 1.676 bytes)

El contenido del archivo Win95io.txt

5.− Funciones para leer/escribir en archivos INI (16 y 32 bits) (1/Mar)

Estas funciones simulan las que incorpora VB4: GetSetting y SaveSetting, pero siempre trabajan con archivos INI, no lo hacen con el registro, como ocurre si el VB4 es 32 bits.

Las funciones usadas del API son: GetPrivateProfileString y WritePrivateProfileString.

En caso de que lo uses con VB3 o anterior, deja sólo la declaración de las funciones del API, sin los

```
#If...#Else...#End If
'------------------------------------------------
' Profile.bas (24/Feb/97)
' Autor: Guillermo Som Cerezo, 1997
' Fecha inicio: 24/Feb/97 04:05
'
' Módulo genérico para las llamadas al API
' usando xxxPrivateProfileString
'------------------------------------------------
Option Explicit
#If Win32 Then
'Declaraciones para 32 bits
Private Declare Function GetPrivateProfileString Lib "Kernel32" Alias "GetPrivateProfileStringA" _
(ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
ByVal lpDefault As String, ByVal lpReturnedString As String, _
ByVal nSize As Long, ByVal lpFileName As String) As Long
Private Declare Function WritePrivateProfileString Lib "Kernel32" Alias "WritePrivateProfileStringA" _
(ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
ByVal lpString As Any, ByVal lpFileName As String) As Long
#Else
'Declaraciones para 16 bits
Private Declare Function GetPrivateProfileString Lib "Kernel" _
(ByVal lpApplicationName As String, ByVal lpKeyName As Any, _
ByVal lpDefault As String, ByVal lpReturnedString As String, _
ByVal nSize As Integer, ByVal lpFileName As String) As Integer
Private Declare Function WritePrivateProfileString Lib "Kernel" _
```

```
(ByVal lpApplicationName As String, ByVal lpKeyName As Any, _

ByVal lpString As Any, ByVal lplFileName As String) As Integer

#End If

'------------------------------------------------------------------------

'Función equivalente a GetSetting de VB4.

'GetSetting En VB4/32bits usa el registro.

' En VB4/16bits usa un archivo de texto.

'Pero al usar las llamadas del API, siempre se escriben en archivos de texto.

'------------------------------------------------------------------------

Public Function LeerIni(lpFileName As String, lpAppName As String, lpKeyName As String, Optional
vDefault) As String

'Los parámetros son:

'lpFileName: La Aplicación (fichero INI)

'lpAppName: La sección que suele estar entrre corchetes

'lpKeyName: Clave

'vDefault: Valor opcional que devolverá

' si no se encuentra la clave.

'

Dim lpString As String

Dim LTmp As Long

Dim sRetVal As String

'Si no se especifica el valor por defecto,

'asignar incialmente una cadena vacía

If IsMissing(vDefault) Then

lpString = ""

Else

lpString = vDefault
```

```
End If
sRetVal = String$(255, 0)
LTmp = GetPrivateProfileString(lpAppName, lpKeyName, lpString, sRetVal, Len(sRetVal), lpFileName)
If LTmp = 0 Then
LeerIni = lpString
Else
LeerIni = Left(sRetVal, LTmp)
End If
End Function
'_____
'Procedimiento equivalente a SaveSetting de VB4.
'SaveSetting En VB4/32bits usa el registro.
' En VB4/16bits usa un archivo de texto.
'Pero al usar las llamadas del API, siempre se escriben en archivos de texto.
'_____
Sub GuardarIni(lpFileName As String, lpAppName As String, lpKeyName As String, lpString As String)
'Guarda los datos de configuración
'Los parámetros son los mismos que en LeerIni
'Siendo lpString el valor a guardar
'
Dim LTmp As Long
LTmp = WritePrivateProfileString(lpAppName, lpKeyName, lpString, lpFileName)
End Sub
```

6.− Desglosar una ruta/nombre de archivo (1/Mar)

Una función para desglosar en el Path y el Nombre del archivo, la ruta que recibe como parámetro.

Creo que está suficientemente explicada, cómo para necesitar más aclaración.

```
Public Sub SplitPath(ByVal sTodo As String, sPath As String, Optional vNombre, Optional vExt)
'--------------------------------------------------------------------
'Divide el nombre recibido en la ruta, nombre y extensión
'(c)Guillermo Som, 1997 ( 1/Mar/97)
'
'Esta rutina aceptará los siguientes parámetros:
'sTodo Valor de entrada con la ruta completa
'Devolverá la información en:
'sPath Ruta completa, incluida la unidad
'vNombre Nombre del archivo incluida la extensión
'vExt Extensión del archivo
'
'Los parámetros opcionales sólo se usarán si se han especificado
'--------------------------------------------------------------------
Dim bNombre As Boolean 'Flag para saber si hay que devolver el nombre
Dim i As Integer
If Not IsMissing(vNombre) Then
bNombre = True
vNombre = sTodo
End If
If Not IsMissing(vExt) Then
vExt = ""
i = InStr(sTodo, ".")
If i Then
vExt = Mid$(sTodo, i + 1)
```

```
End If

End If

sPath = ""

'Asignar el path

For i = Len(sTodo) To 1 Step −1

If Mid$(sTodo, i, 1) = "\" Then

sPath = Left$(sTodo, i − 1)

'Si hay que devolver el nombre

If bNombre Then

vNombre = Mid$(sTodo, i + 1)

End If

Exit For

End If

Next

End Sub
```

11.− Como llamar al Microsoft Internet Mail y News desde un programa VB (5/Mar)

Este "truco" me lo ha enviado Joe LeVasseur

Pon dos botones en un Form e inserta este código:

```
Private Sub Command1_Click()

Dim ValDev&, Programa$

Programa = "EXPLORER.EXE /root,c:\windows\Internet Mail." & _

"{89292102−4755−11cf−9DC2−00AA006C2B84}"

ValDev = Shell(Programa, vbNormalFocus)

End Sub

Private Sub Command2_Click()
```

```
Dim ValDev&, Programa$

Programa = "EXPLORER.EXE /root,c:\windows\Internet News." & _

"{89292103−4755−11cf−9DC2−00AA006C2B84}"

ValDev = Shell(Programa, vbNormalFocus)

End Sub
```

Si usas el Microsoft Internet News/Mail,

se arrancan cuando pulsas el botón.

Es que no hay un EXE para ellos− son hijos del Explorer.

Joe

12.− Ejecutar cualquier archivo, incluso accesos directos (LNK) (13/Mar)

Esta pregunta me había surgido antes y no encontraba la "puñetera" respuesta. Probé con el Explorer.exe, al estilo del truco anterior, pero nada...

De estas cosas que miras la ayuda y "de casualidad" lees que con **start** se pueden ejecutar aplicaciones desde la línea de comando... y si se pueden ejecutar aplicaciones... ¿se podrán ejecutar accesos directos? PUES SI !

Y no sólo accesos directos, sino TODO lo que le eches: archivos de cualquier extensión; el START se encarga de llamar a la aplicación correspondiente... lo que uno se ha complicado haciendo DDE y todo el rollo para esta tarea tan fácil!

¿Cómo se hace?

```
Dim ret As Long

ret = Shell("start " & sFile)

'Si Quieres que no se muestre la ventana:

ret = Shell("start " & sFile, 6)
```

sFile será "lo que queramos" ejecutar. CUALQUIER COSA!

13.− Un Huevo de Pascua (Easter Egg), el del VB4 (24/Mar)

Este "truco" me lo ha mandado el señor Joe LeVasseur y se trata del Easter Egg del Visual Basic 4, se trata de lo siguiente:

Crea un proyecto nuevo e inserta un TextBox, en la propiedad Text escribe: **Thunder**, seleccionalo y marca la opción "lock controls", ahora pasa el cursor por las ToolBox y "sorpresa!"

14.− Ejemplo de cómo restar fechas y horas (26/Mar)

Dos ejemplos de cómo restar fechas y horas.

Para saber los segundos entre dos horas o los días entre dos fechas.

Crea un form con los siguientes controles, dejale los nombre por defecto.

4 TextBox

2 Labels

2 Commands

Distribuyelos para que los dos primeros TextoBoxes estén con el primer label y command, lo mismo con el resto.

Añade lo siguiente al form y pulsa F5

```
'Ejemplo de prueba para restar fechas y horas (26/Mar/97)

'(c) Guillermo Som, 1997

Option Explicit

Private Sub Command1_Click()

Dim t0 As Variant, t1 As Variant

'Text1 Tendrá una fecha anterior

'Text2 tendrá la nueva fecha

t0 = DateValue(Text1)

t1 = DateValue(Text2)

Label1 = t1 − t0

End Sub

Private Sub Command2_Click()

Dim t0 As Variant, t1 As Variant

'Text3 Tendrá una hora anterior
```

```
Text4 = Format(Now, "hh:mm:ss")

t0 = Format(Text3, "hh:mm:ss")

t1 = Format(Text4, "hh:mm:ss")

Label2 = Format(TimeValue(t1) − TimeValue(t0), "hh:mm:ss")

End Sub

Private Sub Form_Load()

'Para probar la diferencia de fechas

Text1 = DateValue(Now)

Text2 = DateValue(Now + 10)

'

'Para probar la diferencia de horas

Text3 = Format(Now, "hh:mm:ss")

Text4 = Format(Now, "hh:mm:ss")

Command1_Click

Command2_Click

End Sub
```

15.− Leer la línea de comandos y quitarle los 'posibles' caracteres de comillas que tenga. (26/Mar)

Algunas veces cuando recibimos un archivo de la línea de comandos, pueden tener caracteres de comillas, sobre todo si trabajamos con VB4 de 32 bits.

Para usar esta función deberás asignarla a una cadena o usarla directamente.

```
sFile = LineaComandos()

Private Function LineaComandos() As String

Dim sTmp As String

Dim i As Integer

'Comprobar si hay algún archivo en la línea de comandos
```

```
sTmp = Trim$(Command$)
If Len(sTmp) Then
'Si tiene los caracteres de comillas, quitarselos
i = InStr(sTmp, Chr$(34))
If i Then
sTmp = Left$(sTmp, i − 1) & Mid$(sTmp, i + 1)
i = InStr(sTmp, Chr$(34))
If i Then
sTmp = Left$(sTmp, i − 1) & Mid$(sTmp, i + 1)
End If
End If
End If
LineaComandos = sTmp
End Function
```

16.− Determinar la Resolución de la pantalla. (10/Abr)

Un truco/colaboración/rutina del colega Joe LeVasseur.

```
Option Explicit
' Como determinar resolución de la
' pantalla con VB4−Win95/NT.
' Dos versiones− con el API y sin...
' Pon tres botones y un textbox encima de
' un form y insertar este codigo.
'
' Joe LeVasseur lvasseur@tiac.net
Private Declare Function GetSystemMetrics Lib "user32" _
```

```
(ByVal nIndex As Long) As Long

Private Sub Command1_Click()

Dim resolucionX&, resolucionY&

resolucionX = GetSystemMetrics(0)

resolucionY = GetSystemMetrics(1)

Text1.Text = CStr(resolucionX & "x" & resolucionY)

End Sub

Private Sub Command2_Click()

Dim resolucionX&, resolucionY&

resolucionX = Screen.Width / Screen.TwipsPerPixelX

resolucionY = Screen.Height / Screen.TwipsPerPixelY

Text1.Text = CStr(resolucionX & "x" & resolucionY)

End Sub

Private Sub Command3_Click()

Text1.Text = ""

End Sub

Private Sub Form_Load()

Text1.Text = ""

Command1.Caption = "&Con API"

Command2.Caption = "&Sin API"

Command3.Caption = "&Borrar"

Me.Caption = "Ejemplo para el Guille"

End Sub
```

17.− Usar tus propias instrucciones en lugar de las de VB. (29/Jun)

Esto no es realmente un truco, es que o lo **adivinas** por **equivocación** o, como en mi caso, lo lees en un libro.

Ya había notado yo cosas raras con algunas variables, pero no me "fijé" en el detalle... en fin, no pretenderás que esté siempre al loro de todo lo que me ocurra... 8−¿

El tema es que si declaras una función con el mismo nombre que una ya existente, se usará esa función o instrucción en lugar de la que incluye el VB.

Por ejemplo, (para seguir siendo un "copión"), pongo el mismo ejemplo que el libro ese que estoy leyendo ahora.

Se trata de una implementación especial de **KILL**, pero en esta nueva versión, permite varios archivos como parámetros

Puedes usarla de esta forma:

```
Kill "archivo1.txt", sUnArchivo$, "archivoX.*"

Kill "UnoSolo.bak"

Function Kill(ParamArray vFiles() As Variant) As Boolean

Dim v As Variant

On Error Resume Next

For Each v In vFiles

VBA.Kill v

Next

Kill = (Err = 0)

End Function
```

El truco está en anteponer **VBA.** a la instrucción propia del VB y así se sabe exactamente a que se está refiriendo.

18.− Descargar una DLL o EXE que esté en memoria (sólo 16 bits) (6/Jul)

Esto puede servir para descargar una aplicación o librería dinámica de la memoria de nuestro Windows. La forma es sencilla, sólo hay que crear un módulo BAS y escribir este código en el SUB MAIN, como parámetro debemos pasarle la DLL o EXE que queremos "eliminar" y este programita se encargará del resto...

AVISO: Esto sólo funcionará de forma correcta en Windows 3.xx **NO USARLO EN WINDOWS 95**.

A mí no me ha funcionado bien en Win95 y deja colgado el Explorer, al menos el que se incluye con el IE 4.0 beta.

El que avisa...

```
'------------------------------------------------------------

'Descargar una DLL o EXE que esté en memoria ( 6/Jul/97)

'

'Basado en un código de Bruce McKinney y que realiza la misma

'tarea que WPS.exe para descargar módulos y ejecutables.

'(se supone)

'------------------------------------------------------------

Option Explicit

Declare Function GetModuleHandle Lib "Kernel" (ByVal lpModuleName As String) As Integer

Declare Function GetModuleUsage Lib "Kernel" (ByVal hModule As Integer) As Integer

Declare Sub FreeModule Lib "Kernel" (ByVal hModule As Integer)

Public Sub Main()

Dim hModule As Integer

'El módulo a librerar se pasa en la línea de comandos

hModule = GetModuleHandle(Command$)

If hModule = 0 Then Exit Sub

'Libera todas copias de este módulo

Do While GetModuleUsage(hModule) > 0

Call FreeModule(hModule)

Loop

End Sub
```

19.− Barra de botones al estilo Office y un ToolTip sencillo (6/Ago)

Esto no es realmente un truco sino más bien una pequeña "utilidad", pero creo que encaja bien en este apartado de los trucos.

Pulsa en este link para ir a la página con la explicación y los listados.

Pulsa en este otro para ver la revisión del 7/Ago/97

21.− No permitir cambiar el tamaño de una ventana redimensionable (31/Ago)

Seguramente te preguntarás ¿que utilidad puede tener esto? Si a la ventana se le puede cambiar el tamaño, ¿por qué no permitir que se cambie?

La respuesta, para mí, es sencilla, pero la dejo para que pienses un poco cual sería el motivo...

Bueno, ahí va: en algunas ocasiones me gusta que los bordes de la ventana se vean de forma "normal", es decir como si se pudiese cambiar el tamaño, pero no me gusta que lo puedan cambiar, así que lo que he hecho en estas ocasiones es simplemente conservar el tamaño inicial de la ventana (el que tiene al cargarse) y cuando el usuario decide cambiarle el tamaño, no permitirselo y volver al que tenía inicialemente.

Este "truco" lo mandé ayer día 30 a la lista de VB−ESP, pero tenía un **inconveniente**: que al cambiar el tamaño por el lado izquierdo o por la parte superior, se movia el form, esto sigue igual, si alguien tiene la forma de conseguirlo, sin que sea dejando el form en la posición inicial, que eso es fácil, sino que **recuerde** la última posición si sólo se ha movido...

Aquí tienes todo el código necesario:

```
'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

'Prueba para no cambiar el tamaño de una ventana con

'bordes dimensionables (30/Ago/97)

'−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

Option Explicit

'Tamaño inicial del Form

Dim iH As Integer

Dim iW As Integer

Private Sub Form_Load()

'Guardar el tamaño inicial

iH = Height

iW = Width

End Sub

Private Sub Form_Resize()
```

```
'Sólo comprobar si el estado es Normal

If WindowState = vbNormal Then

'Si se cambia la altura

If Height <> iH Then

Height = iH

End If

'Si se cambia el ancho

If Width <> iW Then

Width = iW

End If

End If

End Sub
```

Contenido:

Hacer referencia a un control usando una variable

Otro procedimiento para esperar X segundos

Más sobre la colección Forms y Controls (hacer referencia a un control o form usando variables)

Cómo pasar parámetros opcionales de un procedimiento a otro, usando ParamArray. (15/Mar/99)

1.− Posicionar el cursor al final de una línea de texto (4/Sep)

Ya sabes cómo seleccionar todo el texto de un TextBox, ahora puedes usar esto para posicionarte al final:

```
Text1.SetFocus 'Asegurarnos que reciba el foco

Text1.SelStart = Len(Text1) 'La posición del caracter inicial es la longitud del texto...

'por tanto se posiciona al final
```

2.− Acceder a un control por la tecla rápida sin necesidad de pulsar ALT+letra. (21/Sep)

Este "truco" servirá para aquellos forms en los que necesitemos acceder a distintos controles que tienen una tecla de acceso rápido, pero sin necesidad de pulsar la combinación de teclas: Alt+letra_de_acceso.

Para los forms que tengan TextBoxes o cualquier otro control en el que haya que escribir, no se debe usar este truco, ya que impediría introducir esos caracteres, pero como hay ocasiones en las que se puede necesitar, por ejemplo si las entradas del form activo sólo son numéricas o bien si hacemos algo parecido a un MsgBox.

De todas formas, aquí está y si ves que te puede ser útil lo usas y si no, pues "aire" al truco y a otra cosa mariposa...

Este código funciona en cualquier versión de Visual Basic, en la versión 1 y 2 no lo he probado... ¿alguien las usa?

```
Sub Form_KeyPress (KeyAscii As Integer)

'Comprobar si la tecla pulsada coincide con

'alguna de acceso rápido

'

'NOTAS:

' Debe estar puesto Option Compare Text

' El KeyPreview del Form debe estar a True
```

```
' Esto no es demasiado útil si hay TextBoxes
' ya que no podrás escribir los caracteres
' de acceso rápido
' Pero para cualquier otra aplicación está bien
'
Dim ch As String
Dim i%, j%
'Detectar los errores producidos
'al encontrar controles sin Caption
On Local Error Resume Next
ch = Chr$(KeyAscii)
'Un bucle para todos los controles de este form
For i = 0 To Me.Controls.Count − 1
j = InStr(Me.Controls(i).Caption, "&" & ch)
'Si tiene un código de acceso rápido...
If j Then
'Esto es para que descarte la tecla pulsada
KeyAscii = 0
'Enviamos la pulsación Alt+tecla
SendKeys "%" & ch
'nada más que hacer
Exit For
End If
Next
'Si se ha producido un error...
Err = 0
```

'restaurar la rutina de detección de errores

On Local Error GoTo 0

End Sub

3.− Para los que tenemos poca memoria... y VB5 (22/Oct)

Realmente es una chorradilla de truco, pero lo mismo a tí no se te había ocurrido... (la verdad es que a mí tampoco...)

Si estás usando el VB5, sabrás que cuando usas un control, etc., al escribir el punto para poner la propiedad o método a usar, el VB te muestra las posibilidades, hasta aquí estamos de acuerdo, bien..., pues a mi me ocurre que muchas veces no recuerdo los nombres de los controles que tengo en un formulario, ¿que hacía? mostraba el formulario, pulsaba en el control que al que quería hacer referencia y miraba en la ventana de propiedades el nombre... pues hoy se me ocurre, así como el que no quiere la cosa, aunque más bien por probar, a poner Me. y ¡plas! ahí estaban los nombres de todos los controles...

Ya te dije que era una chorrada, pero no sabes lo que me acelera el usar los nombres que les pongo... 8−)

4.− Cómo simular sobreescribir e insertar en un TextBox (12/Ene)

Este truco está sacado de la Microsoft Knowledge Base − How to Emulate Overtype Mode in a Visual Basic Text Box, ID del Artículo: Q96210, por eso los comentarios los he dejado en inglés. Lo único que yo he añadido es el código del evento Text1_KeyDown para que funcione bien al mover el cursor si estamos en modo INSERT.

Este es el código, lo del Label es sólo a título informativo. ¡Que lo disfrutes!

Option Explicit

Const MODE_OVERTYPE = "overtype"

Const MODE_INSERT = "insert"

```
Private Sub Form_Load()

Text1.Tag = MODE_INSERT

Label1.Caption = MODE_INSERT

End Sub

Private Sub Text1_Change()

' You have taken some action that changed the text in the

' text box. Reset the SelLength if you are in overtype mode.

If Text1.Tag = MODE_OVERTYPE And Text1.SelLength = 0 Then

Text1.SelLength = 1

End If

End Sub

Private Sub Text1_KeyDown(KeyCode As Integer, Shift As Integer)

'

'Esto es para manejar bien el movimiento del cursor

'

Select Case KeyCode

' Handle keys that move the caret position and reset the

' SelLength if you are in overtype mode:

Case vbKeyLeft, vbKeyRight, vbKeyUp, vbKeyDown, vbKeyHome, vbKeyEnd, vbKeyPageUp,
vbKeyPageDown

If Text1.Tag = MODE_OVERTYPE Then

Text1.SelLength = 0

End If

End Select

End Sub

Sub Text1_KeyPress(KeyAscii As Integer)

' If you press BACKSPACE and are in overtype mode,
```

```
' then set SelLength to 0 so the backspace will correctly

' delete the character to the left of the current caret

' position. SelLength will be reset when the Text1_Change

' event occurs following the backspace.

If KeyAscii = vbKeyBack And Text1.Tag = MODE_OVERTYPE Then

Text1.SelLength = 0

End If

End Sub

Private Sub Text1_KeyUp(KeyCode As Integer, Shift As Integer)

Select Case KeyCode

' Toggle between insert and overtype modes.

Case vbKeyInsert

If Text1.Tag = MODE_OVERTYPE Then

Text1.Tag = MODE_INSERT

Label1.Caption = MODE_INSERT

Else

Text1.SelLength = 1

Text1.Tag = MODE_OVERTYPE

Label1.Caption = MODE_OVERTYPE

End If

' Handle keys that move the caret position and reset the

' SelLength if you are in overtype mode:

Case vbKeyLeft, vbKeyRight, vbKeyUp, vbKeyDown, vbKeyHome, vbKeyEnd, vbKeyPageUp,
vbKeyPageDown

If Text1.Tag = MODE_OVERTYPE Then

Text1.SelLength = 1

End If
```

```
End Select

End Sub

Private Sub Text1_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)

' You have clicked at a new location within the text box. Reset the

' SelLength if you are in overtype mode.

If Text1.Tag = MODE_OVERTYPE And Text1.SelLength = 0 Then

Text1.SelLength = 1

End If

End Sub
```

5.− Limitar la entrada de un TextBox sólo a números (21/Ene)

Este truco es realmente una colaboración de Esteve, el que está con el gato en la foto de los que dan la cara, yo sólo le he "corregido" un pequeño fallillo que tenía el código que eme envió originalmente...

Realmente la base del truco es el uso de la función IsNumeric, el problema que había era que si se introducía un número decimal menor que 1, había que poner el CERO delante del signo decimal, este caso se resuelve añadiendo ese CERO al valor que se le pasa a esta función... con lo cual acepta cualquier número...

Además, y como regalo extra, se comprueba si se pulsa INTRO y en caso de ser así, se "manda" un TAB, además de evitar el BEEP que se produce al pulsar la tecla Intro.

Si no quieres enviar el TAB, simplemente comenta el **SendKeys "{tab}"** y asunto arreglado...

```
Private Sub Text1_KeyPress(KeyAscii As Integer)

If KeyAscii = 13 Then

KeyAscii = 0 'Para que no "pite"

SendKeys "{tab}" 'Envia una pulsación TAB

ElseIf KeyAscii <> 8 Then 'El 8 es la tecla de borrar (backspace)

'Si después de añadirle la tecla actual no es un número...

If Not IsNumeric("0" & Text1.Text & Chr(KeyAscii)) Then

'... se desecha esa tecla y se avisa de que no es correcta

Beep
```

```
KeyAscii = 0

End If

End If

End Sub
```

6.− Justificar el contenido de un TextBox (22/Feb)

El tema de la justificación del contenido de un textbox es algo simple de solucionar, para ello se debe asignar a la propiedad Multiline el valor True, de esta forma la propiedad Alignment funciona correctamente.

Si sólo quieres que se muestre una línea, como la mayoría de los TextBox normales, el problema surge cuando el usuario pulsa Intro, al ser multiline, permite que se pulse Intro y se desplaza a la siguiente línea...

Para solucionar este pequeño inconveniente, simplemente asigna a 0 el valor de KeyAscii cuando el valor de esta sea 13 ó 10 (Intro o Control+Intro)

7.− Mostrar los elementos de un ComboBox mientra se escribe (4/Abr)

Esto no es nada nuevo, pero es una ampliación de un truco anterior y de una de las colaboraciones.

En esos se hacía con un TextBox y un ListBox, en este caso es sólo con un ComboBox, que tenga la propiedad Style a 0, para que se pueda escribir en él.

Lo que se consigue es que mientras se escriba en el cuadro de texto, se vaya mostrando el item que se parezca más a lo que estamos escribiendo.

Escribe el siguiente código en el form que contenga el Combo:

```
Private Sub Combo1_Change(Index As Integer)

Static YaEstoy As Boolean

On Local Error Resume Next

If Not YaEstoy Then

YaEstoy = True

unCombo_Change Combo1(Index).Text, Combo1(Index)

YaEstoy = False

End If
```

```
Err = 0

End Sub

Private Sub Combo1_KeyDown(Index As Integer, KeyCode As Integer, Shift As Integer)

unCombo_KeyDown KeyCode

End Sub

Private Sub Combo1_KeyPress(Index As Integer, KeyAscii As Integer)

unCombo_KeyPress KeyAscii

End Sub
```

Añade estas declaraciones y procedimientos en un módulo BAS,

(o en el mismo FORM, pero cambia el PUBLIC por PRIVATE):

```
Option Explicit

Dim Combo1Borrado As Boolean

Public Sub unCombo_KeyDown(KeyCode As Integer)

If KeyCode = vbKeyDelete Then

Combo1Borrado = True

Else

Combo1Borrado = False

End If

End Sub

Public Sub unCombo_KeyPress(KeyAscii As Integer)

'si se pulsa Borrar... ignorar la búsqueda al cambiar

If KeyAscii = vbKeyBack Then

Combo1Borrado = True

Else

Combo1Borrado = False

End If
```

```
End Sub

Public Sub unCombo_Change(ByVal sText As String, elCombo As ComboBox)

Dim i As Integer, L As Integer

If Not Combo1Borrado Then

L = Len(sText)

With elCombo

For i = 0 To .ListCount − 1

If StrComp(sText, Left$(.List(i), L), 1) = 0 Then

.ListIndex = i

.Text = .List(.ListIndex)

.SelStart = L

.SelLength = Len(.Text) − .SelStart

Exit For

End If

Next

End With

End If

End Sub
```

9.− Activar la instancia anterior de una aplicación al cargarla por segunda vez (15/May)

Cuando se ejecuta una aplicación de Visual Basic, se puede saber, mediante la propiedad PrevInstance del objeto App, si dicha aplicación se está ejecutando.

El TIP que te traigo hoy es para activar la aplicación que se ejecuto por primera vez.

Es decir sólo quieres que haya una "copia" del programa ejecutándose y si se intenta ejecutar de nuevo, hacer que se "active" la copia que haya en ejecución, en lugar de una nueva.

He de aclarar que este truco sólo sirve si el Caption del programa es siempre el mismo.

Para hacer que se "active" la aplicación incluso si el caption se cambia, por ejemplo en el caso de que sea una

aplicación MDI o que por cualquier razón cambies el Caption tendrás que "ingeniartelas" por tí mismo.

Lo que yo hago en esos casos, es crear una entrada en el registro o en un fichero INI con el caption que tiene mi aplicación cuando éste cambia, de esta forma puedo saber de forma fácil y rápida el nombre que necesito para activar esa instancia del programa.

También se podría averiguar examinando los títulos de las ventanas (aplicaciones) activas y de esta forma activarla, pero eso sería algo más complicado... pero si lo haces, me mandas el código y lo pondría para que otra gente lo viera.

Como pista para conseguirlo, podrías usar el código usado para saber si una aplicación se está ejecutando... ese código está en la utilidad **ListVentanas** que encontrarás en la sección GRATISWARE.

Vamos ya con el código para hacer eso de activar la aplicación que se está ejecutando.

```
Private Sub Form_Load()

Dim sCaption As String

'si ya se está ejecutando

If App.PrevInstance Then

'Guardar el caption de esta aplicación

sCaption = Caption

'Cambiar el caption actual para que no se active esta

Caption = "cualquier cosa"

'Activar la otra instancia

AppActivate sCaption

'Terminar esta copia del programa

End

End If

'Continuar ya que no hay otra copia

End Sub
```

Esto es lo que habría que hacer si el caption de la aplicación cambia y no mantiene siempre el mísmo valor.

Es importante guardar el nuevo caption cada vez que éste se modifique.

```
Private Sub Form_Load()

Dim sCaption As String
```

```
'si ya se está ejecutando

If App.PrevInstance Then

'Leer del fichero de configuración el caption de la aplicación

sCaption = GetSetting("Aplicacion.ini", "General", "Caption", Caption)

'Cambiar el caption actual para que no se active esta

Caption = "cualquier cosa"

'Activar la otra instancia

AppActivate sCaption

'Terminar esta copia del programa

End

End If

'Cuando se cambie el caption de la aplicación,

'guardarlo en el fichero de configuración

SaveSetting "Aplicacion.ini", "General", "Caption", Caption

End Sub
```

También se puede usar este método en el caso de que el inicio de la aplicación esté en un procedimiento SUB MAIN, en ese caso no podrás usar la propiedad Caption en la asignación de sCaption ni es necesario cambiarla para que no se active esta copia, siempre y cuando al iniciarse desde el módulo BAS aún no se haya mostrado el form.

11.− Hacer referencia a un control usando una variable (23/May)

Ya sabes que para asignar un valor de una propiedad de un control debes hacer lo siguiente:

NombreControl.Propiedad = Valor

Por ejemplo para asignar el Caption de Command1, sería:

Command1.Caption = "Nuevo caption"

Pero puede que te encuentres en la necesidad de hacer referencia a un control por medio de una variable, por ejemplo en el caso de que crees una clase que manipule controles pero sólo sabe de ese control el nombre y nada más.

Pues bien, en esos casos, puedes usar la colección de controles que tiene cada Form. Usando el mismo ejemplo de asignar el **Caption** del **Command1**, pero siendo la variable **unControl$** la que tiene el nombre, se puede hacer esto:

Controls(unControl).Caption = "Nuevo Caption"

Si el control está dentro de un array de controles, se tendrá que hacer esto:

Controls(unControl)(Indice).Caption = "Nuevo Caption del indice " & Indice

Esta forma de usar los controles, la tuve que usar en una clase que manipulaba unas etiquetas y unos contenedores, para no obligarme a usar siempre el mismo nombre en las etiquetas y contenedores.

12.− Otra procedimiento para esperar X segundos (28/Ago)

Pues eso, otra forma de esperar un número determinado de segundos.

```
'Si se quiere usar de forma GLOBAL, insertarlo en un Módulo BAS y declararlo como público

Private Sub Wait(ByVal nSec As Integer)

'Esperar un número de segundos

Dim t1 As Date, t2 As Date

t1 = Second(Now)

t2 = t1 + nSec

Do

DoEvents

Loop While t2 > Second(Now)

End Sub
```

13.− Más sobre la colección Forms y Controls (hacer referencia a un control o form usando variables) (11/Oct)

Esto es una ampliación/aclaración sobre el Tip 11, y viene a cuento por unas pruebas hechas en una consulta recibida, que por cierto, se me quedó, como muchas otras en el tintero...

La cuestión es la siguiente:

Modificar propiedades de controles usando una variable tanto para el form como para el control.

Según el Tip 11, se puede referenciar a una propiedad de un control de la siguiente forma:

Controls(nombre_del_control).Propiedad = Valor_de_la_propiedad

También se puede asignar ese control a una variable de tipo Control, para posteriormente referenciar a las diferentes propiedades:

```
'

Dim tControl As Control

Set tControl = Controls(sNombreControl)

tControl.BackColor = 0&
```

Por tanto, se supone que se debería poder hacer esto otro para poder modificar esa misma propiedad:

```
Forms(nombre_form).Controls(nombre_control).Propiedad = valor_propiedad
```

o bien esto otro:

```
'

Dim tForm As Form

Set tForm = Forms(sNombreForm)

tForm.BackColor = vbRed
```

Pues no... al menos a mi no me ha funcionado... me da Type Mismatch (error 13)

El tema de querer hacerlo así, está en poder usar una rutina genérica que permita cambiar algunas propiedades de algunos controles en cualquier form, pero usando variables para indicar esos Forms y esos Controles... (las propiedades deben especificarse "explícitamente", ya que no existe ninguna colección de propiedades).

La solución que he encontrado para hacer esto es la siguiente:

Se busca el nombre del form en cuestión en la colección Forms y se asigna a una variable del tipo Form, después se puede acceder al control indicado usando la colección controls, como se explica un poco más arriba.

Veamos el código de un procedimiento genérico (público) que permite asignar ciertas propiedades... (recuerda que sólo es un ejemplo, así que no me eches en cara que es una chorrada, aunque si tienes el VB6 verás que es muchísimo más simple gracias a CallByName)

```
'

Public Sub Propiedades(ByVal elForm As String, _

ByVal elControl As String, _

ByVal laPropiedad As String, _
```

```
ByVal elValor As Variant)
'Los parámetros se indican como cadena de caracteres,
'salvo el último que indica el valor a asignar
Dim tmpForm As Form
Dim tForm As Form
Dim tControl As Control
'Recorremos la colección Forms en busca del form indicado
For Each tmpForm In Forms
'Si es el mismo nombre, este es el form que queremos
If tmpForm.Name = elForm Then
'Asignarlo a la variable
Set tForm = tmpForm
End If
Next
'Si no se ha encontrado ese form, avisarlo mediante un error
If tForm Is Nothing Then
Err.Raise vbObjectError + 1000, _
"Propiedades", _
"No se ha hallado el form indicado por " & elForm
Else
'Para detectar el error de asignación del control
On Local Error Resume Next
'Asignamos el control deseado a la variable
Set tControl = tForm.Controls(elControl)
If Err Then
Err = 0
```

```
'No atrapar los errores, sino no se mostraría el nuestro...
On Local Error GoTo 0
Err.Raise vbObjectError + 1000, _
"Propiedades", _
"No se ha hallado el control indicado por " & elControl & _
" en el form " & elForm
End If
'interceptamos las propiedades que podemos manipular
'si se deja esto de LCase(laPropiedad), los nombres deben estar en minúsculas
'También puedes usar Option Compare Text en el módulo.
Select Case LCase(laPropiedad)
Case "backcolor"
tControl.BackColor = elValor
Case "forecolor"
tControl.ForeColor = elValor
Case "caption"
tControl.Caption = elValor
Case "text"
tControl.Text = elValor
Case Else
'etc.
End Select
'
'En VB6 se puede usar CallByName para asignar el valor de una propiedad:
'evitandote todo el mogollón de comparaciones...
'CallByName tControl, laPropiedad, VbLet, elValor
```

'

End If

End Sub

'Para usarlo:

Propiedades Me.Name, "Label1", "Caption", "Hola Mundo"

En resumen: si se quiere obtener un "objeto" form usando Forms("nombre del form"), no se puede...

Contenido: Novato = Al_que_empieza

(24/Ene) Bucles For

(24/Ene) Usa siempre Option Explicit

(24/Ene) Hacer comparaciones sin importar que sean mayúsculas o minúsculas

(25/Ene) Evitar que un sub entre en un bucle sin fin...

(15/Feb) Sobre los argumentos con ByVal y ByRef

(15/Feb) Cuidado con las cadenas pasadas al API de Windows con ByVal

(22/Feb) Efecto ToolTip para VB 2.0 y superior

(5/Mar) Comparaciones más rápidas con IF...THEN

(24/Mar) Los declaraciones de Funciones del API y Tipos definidos en un Form o módulo de Clase

(24/Mar) La visibilidad de las variables

(24/Mar) El Tipo de las variables por defecto

(8/Abr) Listados de ejemplo para crear un ToolBar, ToolTips y efectos 3D para VB3

(6/Jul) Evitar que una aplicación se cargue por segunda vez (**VB2 y posteriores**)

(9/Jul) Evitar los eventos en cascada... ¿te suena el OUT OF STACK SPACE?

1.− Bucles For

Me imagino que sabes hacer un bucle For. Lo que voy a contarte es como hacer que funcione más rápido.

Usa variables enteras para el índice

Procura no hacer cálculos dentro del bucle, de valores que no van a cambiar

No indiques la variable del bucle después de Next

No salgas con GOTO, (¿alguien lo usa todavía?), de un bucle For

Listado 1 (correcto, pero...)

```
Dim x, y, b, a$
a$="Hola Mundo"
For x=1 To 10
b = Len(a$)
For y = 1 To b
If Mid$(a$, y, 1) = Chr$(32) Then
GoTo OtraLinea
End If
Next y
OtraLinea:
Next x
```

Listado 2 (pues, eso...)

```
Dim x As Integer, y As Integer, b As Integer, a$
a$="Hola Mundo"
b = Len(a$)
For x=1 To 10
For y = 1 To b
```

```
If Mid$(a$, y, 1) = Chr$(32) Then

Exit For

End If

Next

Next
```

2.− Usa siempre Option Explicit

Acostumbrate a indicar siempre que sea necesario declarar las variables.

Esta opción era una de las pocas cosas que me gustaban de C o Pascal, ya que si no te acostumbras a declar las variables, al final acabas creando muchas más de las que necesitas.

Antes, (léase Qbasic, GWBASIC, etc.), tenía excusa. Ahora no! Ya que puedes obligarte a declarar las variables, si indicas al inicio del código: Option Explicit

Visual Basic puede hacerlo por tí, si se lo indicas en las opciones del proyecto...

Menú Tools, solapa Environment, opción: Require Variable Declaration

La ventaja: que si te equivocas al escribir una variable, ¿a quién no le ha ocurrido?, VB te indicará que no existe.

La desventaja: Que tienes que declarar todas las variables con Dim, etc.

Ya sabes que si no declaras una variable, y por supuesto no tienes la orden Option Explicit, Visual Basic le asigna por defecto el tipo Variant y la crea si no existe. Con los arrays, crea los valores de 0 a 10.

Por tanto puede que incluso estés desperdiciando memoria:

Variant ocupa más espacio de memoria que cualquier otro tipo

Puede que sólo necesites un array de 5 elementos, el resto que no necesitas, está ocupando memoria que puede ser necesaria para otras cosas: velocidad, más variables, etc.

3.− Hacer comparaciones sin importar que sean mayúsculas o minúsculas

Es muy fácil creer, sobre todo para el no iniciado, que "Hola" es lo mismo que "HOLA"

Para VB, o cualquier otro lenguaje, no es así. Estos trabajan comparando carácter por carácter, y la A (a mayúsculas, código ASCII 65), no es igual a la a (a minúscula, código ASCII 97)

Si en a$ esperamos una entrada que sea igual a "Clave", al hacer esta comparación, puede que pienses que estás comprobandolo correctamente:

If a$ = "Clave" Then...

Pero el usuario, puede haber escrito "clave" o "CLAVE", (hay mucha gente que todo lo escribe en mayúsculas...), y la condición no se cumple.

Para "arreglarlo", podemos hacerlo de la siguiente forma:

If Ucase$(a$) = "CLAVE" Then... o

If LCase$(a$) = "clave" Then...

Pero hay otra forma, con la que no tenemos que preocuparnos de cómo se introduzcan los datos, cualquiera de las formas que usemos para comprobarlo, dará un resultado TRUE en la comparación...

¿Cómo?

Indicando OPTION COMPARE TEXT en el inicio del módulo en el que hagamos la comparación.

OJO, sólo a nivel de módulo y en el módulo que se especifique.

De esta forma, el usuario puede escribir "CLAVE" y esta comparación se cumplirá:

If a$ = "Clave" Then...

4.− Evita que al cambiar un ListBox entre en un bucle sin fin.

Para hacer esto, yo uso variables estáticas mientras cambio un ComboBox o un ListBox...

Ya sabes que las variables estáticas conservan el valor entre cada llamada.

A diferencia de las locales−dinámicas, que se generan cada vez que entras en un Sub.

También puedes usar una variable a nivel de módulo, en lugar de una estática.

Ya que la estática es visible sólo por ese Sub, mientras que la variable a nivel de módulo, es visible por todo el módulo, o por toda la aplicación, si está declarada como Pública o Global.

Sub Combo1_Change()

Static YaEstoy as Boolean

If YaEstoy Then Exit Sub

YaEstoy = True

'El código...

```
YaEstoy = False

End Sub
```

5.− Sobre los argumentos con ByVal y ByRef: Argumentos por Valor o por Referencia (15/Feb)

Cuando VB pasa una variable a un procedimiento, lo hace por referencia, es decir, le dá al procedimiento la variable, de modo que cualquier cambio que se haga en dicha variable, se reflejará en el contenido de la misma.

¿Lógico? Se supone, ya que estamos acostumbrados a que así sea, (me refiero a los programadores de Basic). Si queremos pasar sólo el valor de una variable, debemos ponerla entre paréntesis, de esta forma cualquier cambio que el procedimiento realice en la variable, no será "actualizado" al contenido de la misma... Esta bien, veamos unos ejemplos...

Tenemos un sub que recibe dos variables y le suma a la primera el valor de la segunda:

```
Sub Sumar (A As Integer, B As Integer)

A = A + B

End Sub
```

Veamos que ocurre con esto del valor y la referencia:

```
'Parámetros por referencia

M = 5

N = 3

Sumar M, N

Print M 'Imprimirá 8, 5+3

'Parámetros por valor

M = 5

N = 3

Sumar (M), N

Print M 'Imprimirá 5, ya que no se cambia el valor
```

En el segundo ejemplo, la llamada al procedimiento, podríamos haberlo puesto también de esta forma:

Sumar ByVal M, N

El valor de la variable M, al pasarse por valor, no se modifica en el suprograma Sumar, ya que lo que recibe este Sub es una copia con el valor de la variable, no una *referencia* a la dirección de memoria en la que la variable almacena el valor.

Todo esto viene a cuento de que en ocasiones, necesitamos manejar una varibale en un procedimiento, pero no nos interesa que se modifique "accidentalmente" el valor. Este no sería el caso del Sub Sumar, ya que la intención es modificar el valor del primer parámetro. Veamos otro ejemplo, pero esta vez con cadenas de caracteres. (la aclaración de **caracteres** es para el que aún no se ha enterado que normalmente en basic cuando uno se refiere a cadenas, se hace referencia a las cadenas de caracteres, de nada.)

Esta función, que de paso puede sernos de utilidad, recibe dos cadenas, y devolverá el resultado de "extraer" de la primera, el contenido de la segunda; en caso de que la segunda no esté dentro de la primera, se devuelve la primera completa.

Por ejemplo:

Extrae("Hola mundo","mundo") devolvería "Hola " y

Extrae("Hola mundo","gente") devolvería "Hola mundo"

Veamos la declaración de esta función

```
' Listado 1

Public Function Extrae( Cadena As String, Parte As String) As String

Dim i As Integer

i = Instr(Cadena, Parte)

If i Then

Extrae = Left$(Cadena, i−1) & Mid$(Cadena, i + Len(Parte))

Else

Extrae = Cadena

End If

End Function
```

Tal como se manejan los datos en esta declaración, ni Cadena ni Parte se modifican. Pero podríamos tener el código "mal" definido y por "accidente" cambiar estos valores. Vemoslo con una modificación de la función, para que el valor devuelto no tenga espacios delante ni detrás:

```
' Listado 2

Public Function Extrae( Cadena As String, Parte As String) As String

Dim i As Integer
```

```
Cadena = Trim$(Cadena)

i = Instr(Cadena, Parte)

If i Then

Cadena = Trim$(Left(Cadena, i−1)) & Trim$(Mid$(Cadena, i+Len(Parte)))

End If

Extrae = Cadena

End Function
```

Ahora tanto Cadena, como Extrae devolverían lo mismo. Pero puede que nos interese que Cadena siga teniendo el valor original. Esto podemos solventarlo de dos formas:

La primera, usando una variable temporal:

```
' Listado 3

Public Function Extrae( Cadena As String, Parte As String) As String

Dim i As Integer

Dim sTmp As String

sTmp = Trim$(Cadena)

i = Instr(sTmp, Parte)

If i Then

sTmp = Trim$(Left(sTmp, i−1)) & Trim$(Mid$(sTmp, i+Len(Parte)))

End If

Extrae = sTmp

End Function
```

La segunda, cambiado la definición del parámetro Cadena para que sea por valor: en el listado 2, cambiar la definición de la función para que esté de esta forma:

```
Public Function Extrae( ByVal Cadena As String, Parte As String) As String
```

De esta forma, no se reflejará ningún cambio en Cadena y no tendremos que usar otra variable, para hacer el trabajo.

Resumiendo:

ByRef (por Referencia) es la forma por defecto y se pasa la variable, o mejor dicho: se pasa la dirección en

donde se almacenan los datos de esa variable, después verás a que viene esta aclaración.

ByVal pasa sólo el valor.

El siguiente apartado, te muestra las precauciones que debes tener con lo que se ha comentado, cuando se pasan valores a las API de Windows.

6.− Cuidado con las cadenas pasadas al API de Windows con ByVal (15/Feb)

Normalmente las llamadas al API, se hacen por valor, que es la forma que tiene el lenguaje C de hacerlo.

Según lo expuesto en el apartado anterior, si se pasan variables por valor, estamos pasando el valor de la variable y por tanto no se modificará su contenido: **PUES NO...** Bueno, **no** cuando al API de Windows se refiere y si se trata de cadenas.

En el caso de las cadenas pasadas al API con ByVal, se le está pasando la dirección que apunta a los datos, (puntero o referencia a la variable, según los señores de **C**), por tanto la función podrá "libremente" modificar el contenido de la "susodicha" cadena.

Por tanto: precaución. Y para ser precavidos, pues eso: mejor prevenir que curar.

Cuando pasemos cadenas al API de Windows, u otro API cualquiera que use funciones en formato del lenguaje C, debemos asegurarnos que las cadenas son suficientemente largas, para que almacene el valor que deba almacenar.

La precaución: no ser "roñosos" a la hora de declarar la variable.

Por ejemplo, si queremos usar una función que devuelve una cadena, como sería el caso de GetWindowsDirectory, que devuelve el directorio de Windows:

```
Declare Function GetWindowsDirectory Lib "Kernel32" Alias "GetWindowsDirectoryA" _

(ByVal lpBuffer As String, ByVal nSize As Long) As Long

Dim WinDir As String

Dim Cadena As String

Dim ret As Long

Cadena = String$(300, Chr$(0))

ret = GetWindowsDirectory(Cadena, Len(Cadena))

WinDir = Left$(Cadena, ret) 'Esta sería la forma "lógica" de obtener el valor

'Pero podemos "rizar el rizo" y hacerlo de esta otra:

WinDir = Left$(Cadena, Instr(Cadena, Chr$(0)) − 1)
```

Fijate en que he puesto dos formas de asignar a WinDir el directorio de Windows.

La primera es la que se debería hacer, ya que el valor devuelto por la función GetWindowsDirectory es la longitud de la cadena resultante.

Mientras que la segunda es una forma genérica de obtener las cadenas cuando es una función escrita en C la que la devuelve. Y es por la razón de que las cadenas en C, normalmente, siguen el formato ASCIIZ, es decir una cadena acabada en el carácter de código ASCII 0 (cero)

Lo que quiero decir con todo este rollo, es que debemos ser generosos con la longitud de las cadenas pasadas a funciones del API, si sabemos que son directorios o archivos los valores que van a devolver, con una longitud de 260, sería más que suficiente, ya que este valor es el máximo soportado por los nombres largo. Este valor, está definido en la constante MAX_PATH del SDK del API de Windows.

Por cierto en el fichero Win32API.TXT está mal, ya que indica que son 32 caracteres, cualdo deberían ser 260.

7.− Efecto ToolTip para Visual Basic 2.0 y superiores (22/Feb)

Este programa es para hacer el efecto ToolTip.

Adjunto un listado de ejemplo, válido para todos los VB a partir de la versión 2 incluida.

Si quieres echarle un vistazo al listado de la rutina que se encarga de hacer el efecto, pulsa aquí. Está en formato TXT para que puedas verlo directamente con el browser.

Listado de ejemplo en formato zip (tooltip.zip 3.27 KB)

8.− Comparaciones más rápidas con IF...THEN (5/Mar)

Este es un truco "antiguo", pero que aún sirve. Lo he sacado de una revista que tenía de mis tiempos del VIC−20.

El mirar la revista ha sido un poco de añoranza, ya que al comunicarme Álvaro Ibañez de que me había incluido en los recomendados de iWorld, comentó que "Es bueno encontrar gente que *también* conoció los tiempos del Vic−20 & Co... :−)"

Y me puse a hojear las revistas que tenía del Commodore World, que por cierto las tengo todas, soy un "forofo" de las colecciones de revistas que me interesan...

Bueno, vamos al truco en cuestión:

Esto es para cuando se quieren hacer "múltiples" comparaciones usando el AND, por ejemplo:

If A=2 AND B=3 Then ...

Se puede sustituir por esto otro:

If A=2 Then If B=3 Then ...

O si lo hacemos estructurado:

If A=2 Then

If B=3 Then

'...

End If

End If

Como verás, haciendo la comparación con IF en bloques, se entiende mejor el porqué es más rápido de la segunda forma.

Sólo se pasa al segundo IF si el primero se cumple, lo mismo que ocurriría con el AND.

9.− Las declaraciones del API y Tipos definidos en un Form o módulo de Clase (24/Mar)

Cuando declaras una función del API o de una DLL externa, se suele hacer de la siguiente forma:

Declare Nombre_Funcion...

Si pretendes usarla de forma local en un Form o un módulo de Clase, debes ponerle delante Private y así no te dará error.

Private Declare Nombre_Funcion...

Lo mísmo ocurre con los tipos definidos. Antes (VB3 y anteriores), sólo se permitía en los módulos BAS, pero ahora se pueden declarar en los Form y CLS, pero si son privados. Por tanto debes hacerlo con Private delante:

Private Type El_tipo_que_sea...

Por supuesto puedes usar también Private en un módulo BAS, pero sólo estarán visibles dentro de ese módulo.

10.− La "visibilidad" de las variables (24/Mar)

Ya que he comentado lo de las declaraciones en el "consejo" anterior, voy a aclarar un poco de que va eso de la visibilidad de las variables. Y viene al caso porque a más de uno le ha pasado que "se encuentra" atascado porque le da problemas el programilla con cosas "raras".

Veamos las posibilidades que podemos tener al declarar las variables:

Globales: Para usarlas en todo el proyecto. Cuando queramos que una variable sea "visible" a todo el proyecto en el que estamos trabajando, (es decir que podamos usar esa variable en cualquier sitio), debemos declararla como Pública o Global en las declaraciones generales de un módulo BAS.

Globales (Propiedades) Para usarlas en todo el proyecto, pero poniendo el nombre del Form en el que se han declarado delante de la variable, si queremos usarla fuera del propio form. A estas variables se las considera Propiedades del Formulario. Esto es parecido a una variable declarada en un módulo BAS, pero a diferencia de la anterior, debemos especificar el form al que pertenece y la vida de esta variable será mientras exista el form en el que se ha declarado.

Nivel de Módulo: Para usarlas sólo en un módulo o form determinado. Las variables declaradas en la parte de las declaraciones de un módulo, (BAS, FRM o CLS) con el atributo de Private o simplemente con DIM, se consideran locales al módulo y sólo serán visibles dentro de ese módulo.

Nivel de Procedimiento: Para usarlas sólo en un procedimiento. Las que se declaren dentro de un procedimiento (SUB, FUNCTION o PROPERTY) sólo serán visibles dentro de ese procedimiento.

Hay que tener en cuenta que cuando se declara una variable en un nivel inferior, esta declaración "prevalece" sobre cualquier otra, de forma que usará la que tenga más cercana a donde se usa, al menos así ocurre con los procedimientos, en caso de variables a nivel de módulo con respecto a las globales, no lo he probado...

Veamos un ejemplo:

Tenemos un módulo BAS con esta declaración:

```
Option Explicit

Global sVar1 As String
```

En el Form_Load:

```
sVar1= "en Form_Load"

'Mostramos el contenido de sVar1

MsgBox "El contenido de sVar1 es: " & sVar1
```

En un procedimiento tenemos:

```
Private Sub Form_Click()

Dim sVar1 As String

sVar1 = "en Procedimiento"

'Mostramos el contenido de sVar1

MsgBox "El contenido de sVar1 es: " & sVar1

End Sub
```

Bien prueba este ejemplo y verás lo que ocurre. Para comprobarlo, haz CLICK en el form.

11.− El Tipo de las variables por defecto (24/Mar)

En los tiempos del GWBASIC y otros Basics, antes del Visual Basic versión 2.0, las variables que no se definían de un tipo en especial, eran variables tipo Single. A partir de la versión 2.0 de VB se introdujo el tipo Variant y de esta forma si la variable no se declara de un tipo en especial, pues VB entiende que son del tipo Variant. En aquellos tiempos (los del Basic normal), era una costumbre más o menos extendida, al menos para los que nos gustaba "ahorrar" memoria y ganar en velocidad, poner al principio del módulo la siguiente instrucción: Defint A−Z, con ella declarabamos todas las variables sin un tipo específico como variables INTEGER. Pues hoy he recibido un mail de Harvey Triana para que lo recomendara a los "novatos" y así lo hago.

Si sueles usar, por regla general, variables de tipo entero, acostumbrate a incluir al principio de cada módulo (Frm, Bas o Cls) las siguientes declaraciones:

Option Explicit

Defint A−Z

De esta forma te obligará a declarar todas las variables (Option Explicit) y a las que no le des un tipo en particular, las tomará por Integer (Defint A−Z). ¡**OJO CON ESTO**! si estás acostumbrado a que las variables por defecto sean Variant, sobre todo en las que declares como **opcionales**, que **siempre deben ser Variant**, al menos en la versión 4, ya que VB5 permite especificar el tipo.

12.− Listados de ejemplo para crear un ToolBar, ToolTips y efectos 3D para VB3 (8/Abr)

Estos listados son un ejemplo para crear un ToolBar, incluye también la rutina para hacer el efecto de los ToolTips y hacer el efecto 3D en cualquier control.

Estos listados són válidos para VB3 (y creo que VB2)

Listados de ejemplo para crear el ToolBar, los ToolTips y ejemplo para efectos 3D (tb_vb3.zip 5.67 KB)

Aclaración: El hecho de que esté aquí es porque en su día puse en este apartado lo de los ToolTips.

13.− Evitar que una aplicación se cargue más de una vez (VB2 y posteriores) (6/Jul)

Rectificación (11/Jul): Por error u omisión... vamos, por despiste, indicaba que el PrevInstance no servía para VB3 y si que funciona; es con el VB2 (¿y anterior?) con el que hay que usar el API para impedir que una aplicación se cargue más de una vez. **Gracias a Arturo Tena por la aclaración** en las NEWS, pero es que algunas veces no prueba uno todo lo que dice...

Algunas veces nos puede interesar que nuestra aplicación sólo se ejecute una vez, pues bien, con el VB3 y superiores eso es fácil, ya que existe una instrucción que ahora veremos, pero en VB2, la cosa no es tan sencilla, aquí te muestro cómo hacerlo para todas las versiones.

Para VB3 y posteriores usa la propiedad PrevInstance del objeto App. Sólo tienes que poner esto en el Form_Load:

```
If App.PrevInstance Then

End

End If
```

En el caso de VB2, podemos usar unas funciones del API de Windows: GetModuleHandle y GetModuleUsage.

Donde dice "nombre.exe" deberás usar el nombre que tenga la aplicación.

Un aviso: Esto sólo debes usarlo en Win3.x aunque en Win95 funcione... debes usarlo con prudencia...

```
'Poner estas declaraciones en un módulo BAS

Declare Function GetModuleHandle Lib "Kernel" (ByVal lpModuleName As String) As Integer

Declare Function GetModuleUsage Lib "Kernel" (ByVal hModule As Integer) As Integer

'En el Form de inicio, poner esto en el Form_Load:

Private Sub Form_Load()

Dim hModule As Integer

Dim numCopias As Integer

hModule = GetModuleHandle("nombre.exe")

numCopias = GetModuleUsage(hModule)

If numCopias > 1 Then

End

End If

'...

End Sub
```

14.− Evitar los eventos en cascada... ¿te suena OUT OF STACK SPACE? (9/Jul)

Sé que no es cosa nueva, incluso el 4º truco de esta sección trata de lo mismo, pero al parecer alguno no captasteis la intención de la solución allí presentada. Aprovecho una de las consultas hechas para poner la respuesta, que de camino sirve un poco de explicación. Allá va... Y si aún no te enteras... pregunta, pregunta.

> una: que es el error "cascada de eventos"

Esto es un error que nos suele pasar a todos cuando empezamos con el

VB, ya que el Windows está enfocado en los eventos, es decir que si

pulsas sobre un TextBox con el ratón, se producen una serie de

eventos, si pulsas una tecla en un control se producen otra serie de

eventos... total que hasta si te rascas la cabeza se produce un

evento... y si mientras estás escribiendo en un TextBox haces algo

que obligue a que se produzca ese evento de nuevo... produces una

cascada de eventos...

Por ejemplo prueba esto:

```
Private Sub Text1_Change()
Dim sTmp As String
sTmp = Text1.Text
Text1.Text= sTmp & "a"
sTmp =""
End Sub
```

Esto te producirá un error de que no hay espacio en la pila o algo

similar, ya que cada vez que se produce el "evento" Change, haces que

cambie el contenido del Text, por lo que vuelve a producirse un nuevo

evento change, sin que termine el anterior... y el VB se pone "malo"

Solución: evitar este tipo de cosas... (elemental)

¿Cómo? con trucos y variables estáticas, por ejemplo:

```
Private Sub Text1_Change()
Dim sTmp As String
```

```
Static YaEstoy As Integer

If YaEstoy Then Exit Sub

YaEstoy = TRUE

sTmp = Text1.Text

Text1.Text= sTmp & "a"

sTmp =""

YaEstoy = FALSE

End Sub
```

Al crear una variable estática mantiene el valor entre llamadas.

La primera vez que entra en el evento, al valer CERO (FALSE), pasa

por alto el IF YAESTOY THEN...

Por tanto se pasa al resto del código y se asigna un valor TRUE

Ahora al asignar el valor al Text1, se produce el evento change, pero

cuando entra por segunda vez, se encuentra con que la condición es

cierta, por tanto se sale sin continuar...

Una vez que termina el primer evento, el valor YaEstoy vuelve a

ponerse a CERO para que en el siguiente se vuelva a procesar lo que

haya que procesar...

Novato = Al_que_empieza (24/Ene)

Que nadie se mosquee. no es por faltar al respeto; pero el que se inicia, siempre es un novato; aunque algunos que llevamos algunos años, puede que sepamos menos, pero... así es la vida, los novatos son los que se inician y los veteranos, aunque sepan igual o menos, son lo que llevan más tiempo... 8−)

En fin, esta página va dedicada a aquellos que necesitan saberlo todo desde el principio.

A ver si os ponéis pronto al dia en la programación en VB.

¿Por qué te das por aludido Manolo? 8−))))

ir al