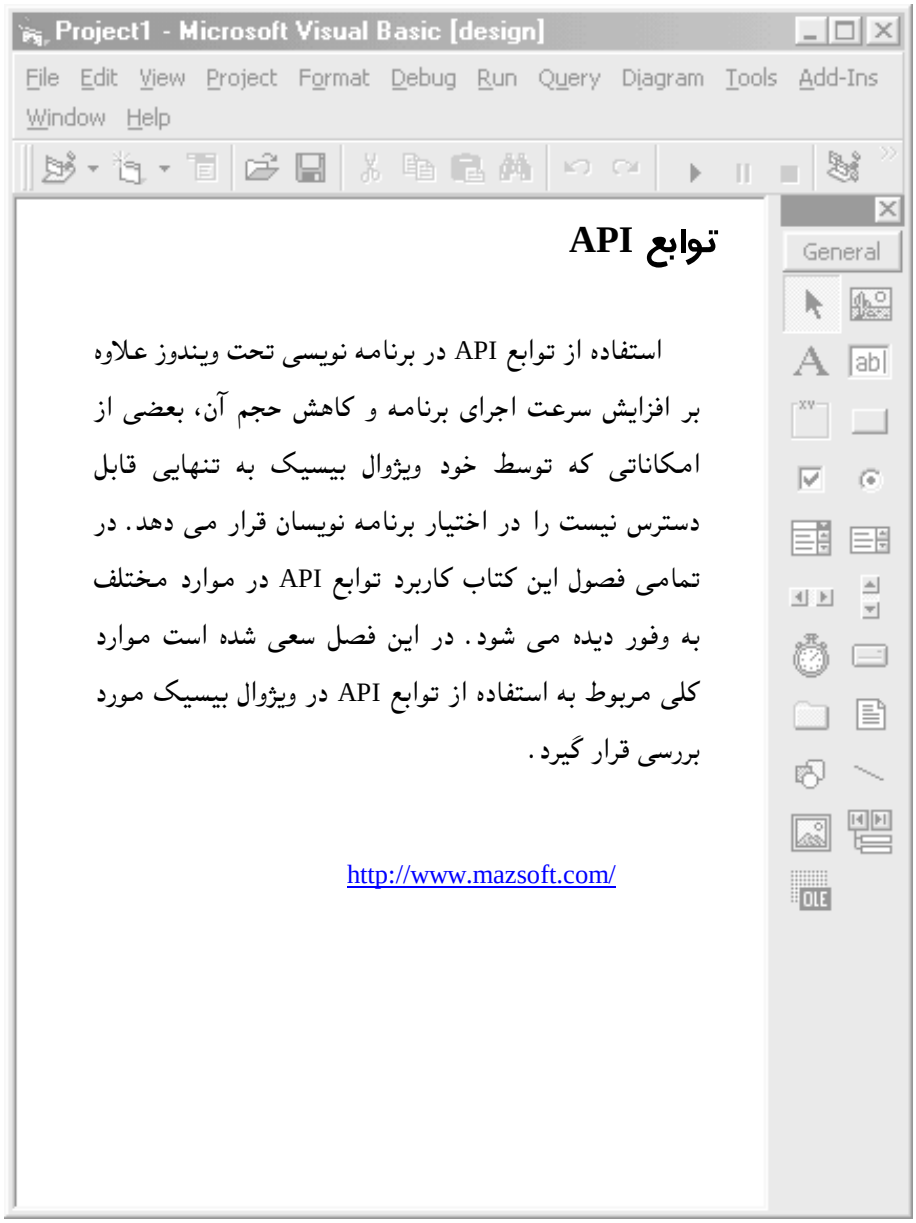
Project1 - Microsoft Visual Basic [design]
File Edit View Project Format Debug Run Query Diagram Tools Add-Ins
Window Help
General
توابع API
استفاده از توابع API در برنامه نویسی تحت ویندوز علاوه بر افزایش سرعت اجرای برنامه و کاهش حجم آن، بعضی از امکاناتی که توسط خود ویژوال بیسیک به تنهایی قابل دسترس نیست را در اختیار برنامه نویسان قرار می دهد. در تمامی فصول این کتاب کاربرد توابع API در موارد مختلف به وفور دیده می شود. در این فصل سعی شده است موارد کلی مربوط به استفاده از توابع API در ویژوال بیسیک مورد بررسی قرار گیرد.
http://www.mazsoft.com/

برنامه نویسی به کمک توابع API[1]

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
190000.TXT	۷ آگوست ۱۹۹۸	4 , 5 , 6	95 / 98 / NT

در این بخش روش کلی استفاده از توابع API در برنامه های ویژوال بیسیک را مورد بررسی قرار خواهیم داد. به این منظور فرض بر این خواهد بود که شما با برنامه نویسی با ویژوال بیسیک آشنا هستید اما تا کنون از ویژوال بیسیک برای فراخوانی توابع API استفاده نکرده اید.

گرچه ویژوال بیسیک سری کامل و ارزشمندی از دستورات، روشها، واقعه ها و روتین ها را برای برنامه نویسی سریع و آسان در اختیار می گذارد، اما محدودیت هایی در محیط برنامه سازی آن وجود دارد که ممکنست شرایط پیاده سازی ایده هایتان را فراهم نکند. یکی از روشهای غلبه بر این محدودیت ها فراخوانی توابع API از درون برنامه های ویژوال بیسیک است. در قلب سیستم عامل ویندوز توابعی وجود دارند که API ویندوز را تشکیل می دهند. بیشتر این توابع از طریق برنامه های ویژوال بیسیک قابل استفاده هستند.

اگر بخواهید کاری انجام دهید که از محدوده توانایی ویژوال بیسیک خارج باشد، باید از توابع API استفاده کنید. مثلا، از طریق هیچ یک از دستورات، خصوصیات یا روتین های ویژوال بیسیک نمی توان سرعت چشمک زدن مکان نمای متن را تغییر داد، ولی می توانید با استفاده از تابع GetCaretBlinkTime سرعت فعلی چشمک زدن مکان نمای متن را بر حسب هزارم ثانیه پیدا کنید و اگر در شرایط خاصی بخواهید سرعت چشمک زدن مکان نمای متن را تغییر دهید، می توانید تابع SetCaretBlinkTime را صدا بزنید و مقدار زمان بین هر دو چشمک را بر حسب هزارم ثانیه برایش ارسال کنید.

تقریبا تمامی خواص ویژوال بیسیک را می توانید با فراخوانی توابع API مربوطه بکار گیرید. البته استفاده مستقیم از این توابع معمولاً پیچیده است و اگر درست انجام نشود

[1] Application Programming Interface

می تواند باعث بروز رفتارهای غیر قابل پیش بینی و احیاناً ایجاد مشکل در سیستم شود. برای کسب بهترین نتیجه باید از امکانات داخلی خود ویژوال بیسیک برای انجام کارهای معمولی استفاده کنید.

چگونگی فراخوانی توابع API

برای فراخوانی توابع API دو مرحله باید طی شود:

- تابع را در یک ماژول[۲] یا فرم تعریف کنید.
- تابع را مانند توابع دیگر در برنامه تان صدا بزنید.

سرفصل "تعریف یک روتین DLL" در "قسمت ۴ – دسترسی به DLL ها و توابع API" از "راهنمای استفاده از ابزارهای ویژوال بیسیک" روش تعریف یک تابع در یک ماژول را بیان می کند. به علاوه، این سرفصل نشان می دهد چگونه از API Text Viewer برای یافتن تعریف یک تابع خاص و کپی کردن آن در برنامه خود استفاده کنید.

برای مثال، برای تعریف تابع GetCaretBlinkTime در یک ماژول، متن زیر را در ماژول خود وارد کنید:

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

اگر می خواهید از این تابع فقط در یک فرم خاص استفاده کنید، کلمه Private را به ابتدای تعریف تابع اضافه کنید. به این ترتیب تعریف تابع GetCaretBlinkTime بصورت زیر در فرم مربوطه قرار می گیرد:

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

این تابع مقداری از نوع Long را بر می گرداند که نشان دهنده سرعت چشمک زدن مکان نما بر حسب هزارم ثانیه خواهد بود. عبارت Lib قبل از نام فایلی که حاوی تابع مورد نظر است قرار می گیرد، در این مورد نام فایل، User32.DLL است. عبارت Alias برای تعریف نام تابع در

---

[۲] Module

فایل DLL بکار می رود. زمانی که نام تابع در برنامه عیناً مانند نام تابع در داخل فایل DLL تعریف شود، استفاده از عبارت Alias اختیاری است.

اغلب در مثالهای بانک اطلاعاتی مایکروسافت، دیده می شود که عبارت Private بکار رفته است و دلیل آن اینست که استفاده نکردن از ماژولها باعث ساده تر شدن مثال می گردد. اگر نیاز دارید از یک تابع API در بیش از یک فرم استفاده کنید باید آنرا در یک ماژول تعریف کنید.

برای کسب اطلاعات بیشتر در مورد دستور Declare ، بخش "Declare Statement" را در Books Online مطالعه نمایید.

پس از اینکه تابع را تعریف نمودید، می توانید آنرا مانند توابع ویژوال بیسیک فرابخوانید و پارامترهای لازم را برایش ارسال نمایید. مثال زیر نشان می دهد چگونه تابع GetCaretBlinkTime را فرابخوانید:

```
Dim lngCaretBlinkTime as Long   ' Blink Rate variable
lngCaretBlinkTime =  GetCaretBlinkTime
```

در مورد توابعی که به پارامترهایی نیاز دارند، می توانید پارامترها را بصورت ارجاعی[۳] یا مقداری[۴] ارسال نمایید.

پارامتری که بصورت ارجاعی ارسال می شود، در اصل آدرس ۳۲ بیتی محل ذخیره مقدار را برای تابع ارسال می کند. ویژوال بیسیک بطور پیش فرض پارامترها را بصورت ارجاعی ارسال می کند. به علاوه می توانید برای تأکید بر ارسال پارامترها بصورت ارجاعی از عبارت Byref استفاده کنید.

مقدار پارامتری که بصورت ارجاعی ارسال می شود می تواند توسط روتینی که صدا زده شده است تغییر یابد. برای مثال، تابع InvertRect یک محدوده مستطیل شکل مشخص را با معکوس کردن تک تک پیکسل ها، بطور کامل معکوس می کند. پارامترهای مورد نیاز این تابع عبارتند از شناسه دستگاه حاوی محدوده مورد نظر و نیز آدرس یک متغیر تعریف شده

---

[۳] By Reference

[۴] By Value

توسط کاربر که حاوی مختصات گوشه های محدوده باشد. آدرس متغیر تعریف شده توسط کاربر بصورت ارجاعی ارسال می شود. هنگامی که این تابع را صدا می زنید، مختصات محدوده مستطیل شکل، در متغیر تعریف شده توسط کاربر قرار می گیرند. برای استفاده از این تابع باید، تعریف های زیر در یک ماژول قرار گیرند:

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

Declare Function InvertRect Lib "user32" Alias "InvertRect" _
(ByVal hdc As Long, ByRef lpRect As RECT) As Long
```

تعریف اول، یک متغیر تعریف شده توسط کاربر بوجود می آورد که مورد نیاز تابع InvertRect است را بوجود می آورد. تعریف دوم خود تابع را تعریف می کند.

ارسال پارامتر بصورت مقداری باعث می شود خود مقدار پارامتر برای تابع ارسال شود نه آدرس محل ذخیره آن. زمانی از این نوع ارسال پارامتر استفاده می کنیم که تابع صدا زده شده تغییری در مقدار پارامتر بوجود نمی آورد. برای ارسال یک پارامتر بصورت مقداری، عبارت ByVal را به آن پارامتر اضافه می کنیم.

☜ توجه: هنگامی که یک متغیر رشته ای را برای یک تابع API ارسال می کنید، عملاً آدرس حافظه محل ذخیره رشته ارسال می شود. بنابراین همیشه باید پارامترهای رشته ای را بصورت ByVal ارسال نمایید. اگر یک پارامتر رشته ای را بصورت ارجاعی ارسال کنید، در اصل آدرس بخشی از حافظه که حاوی آدرس حافظه محل ذخیره رشته است را ارسال نموده اید، و این کار باعث می شود تابع API که این پارامتر برایش ارسال شده، بصورت نادرست عمل کرده و حتی باعث بهم ریختن حافظه شود.

تابع SetCaretBlinkTime به یک پارامتر نیاز دارد که همان سرعت چشمک زدن مکان نما بر حسب هزارم ثانیه است. این پارامتر باید بصورت مقداری برای پارامتر ارسال شود. برای تعریف این تابع در یک ماژول از دستور زیر استفاده نمایید:

```
Declare Function SetCaretBlinkTime Lib "user32" Alias _
"SetCaretBlinkTime" (ByVal wMSeconds As Long) As Long
```

در مثال زیر، سرعت چشمک زدن مکان نما ۲۵۰ هزارم ثانیه تنظیم می شود:

```
Dim lBlinkRate as Long   ' Blink rate variable
Dim lResult as Long      ' Result Variable
lBlinkRate = 250
lResult = SetCaretBlinkTime (lBlinkRate)
If lResult = 0 Then
   ' Error code
End If
```

پروژه نمونه

در این بخش طی مثالی خواهیم دید که چگونه می توان با فراخوانی دو تابع از توابع API سرعت چشمک زدن مکان نما را کنترل کرد. تابع GetCaretBlinkRate سرعت فعلی چشمک زدن مکان نما را بر حسب هزارم ثانیه مشخص می کند، در صورتیکه تابع SetCaretBlinkRate سرعت چشمک زدن مکان نما را تغییر می دهد. هر دو این توابع در یک ماژول تعریف خواهند شد. در صورت نیاز می توانید با اضافه کردن عبارت Private به ابتدای تعریف تابع، عمل تعریف تابع را در یک فرم انجام دهید.

☜ توجه: تابع SetCaretBlinkRate سرعت چشمک زدن مکان نما را برای کل سیستم عامل ویندوز تغییر می دهد. اگر پیش از خروج از برنامه این سرعت را به حال اول باز نگردانید، تغییرات داده شده روی تمامی برنامه ها تاثیر خواهد گذارد.

۱- یک پروژه معمولی در ویژوال بیسیک ایجاد کنید. Form1 بطور پیش فرض بوجود می آید.

۲- یک ماژول به پروژه اضافه کنید. این کار بصورت زیر انجام می گیرد:

- از منوی Project گزینه Add module را انتخاب کنید. پنجره Add module ظاهر می شود.
- از تابلوی New آیکون Module را انتخاب کرده، OK را کلیک کنید. یک ماژول جدید به نام Module1 به پروژه شما اضافه خواهد شد.

۳- کد زیر را در Module1 وارد کنید:

```
Option Explicit

' Returns the current cursor blink rate
Declare Function GetCaretBlinkTime Lib "user32" () As Long
```

```
' Sets the cursor blink rate
Declare Function SetCaretBlinkTime Lib "user32" _
      (ByVal wMSeconds As Long) As Long

' Returns the error code if either GetCaretBlinkTime or
' SetCaretBlinkTime functions fails
Public Declare Function GetLastError Lib "kernel32" () As Long
```

۴- یک HscrollBar، یک TextBox و یک CommandButton به Form1 اضافه کنید.

۵- کد زیر را به Form1 اضافه کنید:

```
Option Explicit
Dim lDefaultTime As Long
Dim lResult As Long
Dim lErrorCode As Long

Private Sub Form_Load()
   lResult = GetCaretBlinkTime
   If lResult = 0 Then
      Call DisplayError(0)
   Else
      lDefaultTime = lResult
      HScroll1.Min = 10
      HScroll1.Max = 1000
      HScroll1.Value = lDefaultTime
      Text1.Text = CStr(lDefaultTime)
      Command1.Caption = "Return to Default"
   End If
End Sub

Private Sub Command1_Click()
   lResult = SetCaretBlinkTime(lDefaultTime)
   ' If the function fails then display
   ' a message box with the error code
   If lResult = 0 Then
      Call DisplayError(1)
   Else
      ' Display the new blink rate.
      HScroll1.Value = lDefaultTime
      Text1.Text = CStr(GetCaretBlinkTime)
      HScroll1.SetFocus
   End If
End Sub

Private Sub HScroll1_Change()
   lResult = SetCaretBlinkTime(HScroll1.Value)

   If lResult = 0 Then
      Call DisplayError(1)
   Else
      lResult = GetCaretBlinkTime
      If GetCaretBlinkTime = 0 Then
         Call DisplayError(0)
      Else
         Text1.Text = CStr(lResult)
      End If
   End If
```

```
End Sub

Private Sub HScroll1_Scroll()
   Text1.Text = CStr(HScroll1.Value)
End Sub

Private Sub Text1_KeyPress(KeyAscii As Integer)
   If KeyAscii = vbKeyReturn Then
      Dim iTextValue As Integer
      iTextValue = CInt(Text1.Text)

      If iTextValue > 1000 Or iTextValue < 10 Then
         MsgBox "Enter a number between 10 and 1000."
         Text1.Text = CStr(HScroll1.Value)
         Exit Sub
      Else
         HScroll1.Value = iTextValue
      End If
   End If
End Sub

Private Sub Form_Terminate()
   lResult = SetCaretBlinkTime(lDefaultTime)

   If lResult = 0 Then
      lErrorCode = GetLastError
      MsgBox ("SetCaretBlinkTime failed. Error" & _
               CStr(lErrorCode))
   End If
End Sub

Private Sub DisplayError(iFail As Integer)
   Dim sErrorMsg As String
   lErrorCode = GetLastError

   Select Case iFail
      Case 0   ' GetCaretBlinkRate Function Failed
         sErrorMsg = "GetCaretBlinkRate Failed. Error Code "
         sErrorMsg = sErrorMsg & CStr(lErrorCode)
      Case 1
         sErrorMsg = "SetCaretBlinkRate Failed. Error Code "
         sErrorMsg = sErrorMsg & CStr(lErrorCode)
      Case Else
         sErrorMsg = "Unknown Error"
   End Select

   MsgBox (sErrorMsg)
End Sub
```

۶- از منوی Run، گزینه Start را انتخاب کرده یا کلید F5 را بزنید تا برنامه اجرا شود. Scroll bar را جابجا کنید تا سرعت چشمک زدن مکان نما تغییر کند. به علاوه می توانید عددی بین ۱۰ تا ۱۰۰۰ را در TextBox وارد کنید تا سرعت چشمک زدن مکان نما تغییر کند. روی دکمه Return to Default کلیک کنید تا سرعت چشمک زدن مکان نما به حالت

اولیه بازگردد. برای خروج از برنامه به جای استفاده از دکمه پایان برنامه که در محیط ویژوال بیسیک وجود دارد، باید روی دکمه بستن پنجره که در انتهای سمت راست نوار عنوان قرار دارد، کلیک کنید. به این ترتیب هنگام خروج از برنامه سرعت چشمک زدن مکان نما به وضعیت اولیه باز خواهد گشت.

نکات

۱- زود به زود برنامه خود را ذخیره کنید. از آنجا که در حال استفاده از توابع API هستید سیستم کنترل اشکالات ویژوال بیسیک نمی تواند اشکالات برنامه را در مورد نحوه استفاده از توابع API به شما گوشزد کند. اگر تابعی درست اجرا نشود، ممکنست پیام خطای سیستم ایجاد شود و کامپیوتر شما را متوقف نماید. برای اطمینان می توانید از ویژوال بیسیک بخواهید همیشه قبل از اجرا کردن برنامه، ابتدا برنامه شما را ذخیره نماید. برای این کار باید گزینه Prompt to save changes از تابلوی Environment پنجره Options را فعال نمایید. برای دیدن این پنجره باید از منوی Tools گزینه Options را انتخاب کنید.

۲- به جای استفاده از متغیرهای آزاد از نوع Any ، نوع متغیرها را بطور صریح مشخص کنید. تعدادی از توابع معرفی شده در API Text Viewer می توانند همه نوع متغیری را به عنوان پارامتر بپذیرند. این توابع، پارامترهایی از نوع Any در لیست پارامترهای خود دارند. برای اینکه رفع اشکال برنامه خود را ساده تر کنید بهتر است در تعریف تابع به جای Any ، نوع متغیری که قرار است استفاده شود را مشخص کنید.

۳- کنترل کنید که هنگام فراخوانی توابع، نوع متغیرها، ثابت ها و مقادیری که برای تابع ارسال می کنید، با آنچه تابع به آن نیاز دارد همخوانی داشته باشد. برای انجام این کنترل از مستندات تابع که در SDK پلاتفرم ذکر شده است استفاده کنید. SDK برای برنامه نویسان C نوشته شده و از نوع داده های C استفاده می کند، بنابراین باید برای تبدیل نوع داده های C به ویژوال بیسیک از جدول زیر استفاده کنید:

معادل ویژوال بیسیک	نوع داده بکار رفته در تابع (زبان C)
ByVal Long	Int , INT
ByVal Long	UINT
ByVal Long	BOOL
ByVal Integer	WORD
ByVal Long	DWORD
ByVal Long	WPARAM
ByVal Long	LPARAM , LRESULT
ByVal Long	COLORREF
ByVal Integer	ATOM
ByVal Long	HANDLE
ByVal Byte	BYTE
ByVal Byte	Char
ByRef Long	LPINT, int *
ByRef Long	LPUINT, UINT *
ByRef Long	LPBOOL, BOOL *
ByRef Byte	LPBYTE, BYTE *
ByRef Integer	LPWORD, WORD *
ByRef Long	LPDWORD, DWORD *
ByRef Long	LPHANDLE, HANDLE *

۴- با استفاده از Add-In Manager برنامه API Text Viewer را به عنوان ابزار Add-in به محیط برنامه نویسی ویژوال بیسیک اضافه کنید. برای کسب اطلاعات بیشتر در زمینه Add-in Manager به بخش Using Wizards and Add-Ins در " Chapter 4-Managing Projects" از کتاب "Visual Basic Programmer's Guide" رجوع کنید.

۵- هنگام استفاده از API Text Viewer، فایل متنی API را به یک بانک داده Access تبدیل کنید تا عمل جستجو سریعتر انجام شود.

٦- همیشه هنگام ارسال رشته ها به یک DLL از عبارت ByVal استفاده کنید.

مراجع

در صورتیکه مایل باشید اطلاعات بیشتری در زمینه کاربرد توابع API در ویژوال بیسیک بدست آورید می توانید به مراجع زیر مراجعه کنید:

- VB5DLL.DOC : این فایل که در دایرکتوری TOOLS\DOCS از CD ویژوال بیسیک قرار دارد، حاوی نکاتی در مورد ساختن کتابخانه های DLL برای استفاده در ویژوال

بیسیک می باشد. برای اینکه بدانید یک DLL از ویژوال بیسیک چه انتظاراتی دارد، این متن را مطالعه نمایید.

- کتاب Visual Basic 5.0 Programmer s Guide to Win32 API نوشته Dan Appleman. استفاده از این کتاب برای برنامه نویسان ویژوال بیسیک که می خواهند از API استفاده کنند بسیار مفید است. این کتاب حاوی اطلاعاتی در مورد استفاده از API است که مخصوص برنامه نویسان ویژوال بیسیک نوشته شده است.
- کتاب Hardcore Visual Basic نوشته Bruce McKinnely حاوی مثالهای خوبی است از نحوه گسترش توانایی های ویژوال بیسیک به کمک API.
- مشترک شبکه برنامه سازان مایکروسافت شوید. این شبکه حاوی فایلهای header استفاده شده برای کامپایل کردن DLL های مورد استفاده سیستم عامل است. از این اطلاعات می توانید برای کنترل درستی توابع، نوع متغیر ها و مقادیر ثابتهای مورد استفاده در برنامه خود استفاده کنید.
- فصل ۵ از کتاب Mastering Visual Basic 5 تحت عنوان Dynamic Link Libraries حاوی خودآموز خوبی در مورد استفاده از API است.

فراخوانی نگارش Unicode توابع API توسط ویژوال بیسیک

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
145727.TXT	۷ آگوست ۱۹۹۸	4 , 5 , 6	95 / 98 / NT

تعریف کردن نگارش Unicode توابع API و فراخوانی آنها به سبکی که در مورد نگارش ANSI آنها عمل می کنیم کافی نیست و نتیجه نمی دهد. در این بخش خواهیم دید که چگونه

بدون نیاز به ایجاد یک کتابخانه نشانه[5]، نگارش Unicode یک تابع API را مورد استفاده قرار دهیم.

ویژوال بیسیک نیز مانند ویندوز NT از لحاظ داخلی بر اساس استاندارد Unicode دو-بایتی بنا نهاده شده است. با این حال ویژوال بیسیک فرض را بر این می گذارد که جهان بیرون از خودش هنوز از حالت تک بایتی ANSI استفاده می کند. هر رشته حرفی که بصورت پارامتر به یک تابع خارجی ارسال می شود پیش از فراخوانی تابع توسط ویژوال بیسیک از حالت Unicode داخلی بصورت ANSI تبدیل می شود. به همین ترتیب، فرض می شود که تمام رشته های حرفی که از یک تابع خارجی به ویژوال بیسیک برگردانده می شوند ANSI هستند و ویژوال بیسیک سعی خواهد کرد برای استفاده داخلی خودش آنها را به صورت Unicode تبدیل نماید. این نحوه عمل باعث می شود فراخوانی توابعی که با پارامترهای حرفی Unicode کار می کنند نسبت به توابعی که با پارامترهای حرفی ANSI کار می کنند متفاوت باشد.

یک راه برای فراخوانی توابع Unicode ایجاد یک کتابخانه نشانه برای تابع Unicode مورد نظر است. سپس باید از درون ویژوال بیسیک به این کتابخانه رجوع کرد و تمامی فراخوانی های تابع را از طریق این کتابخانه انجام داد.

برای کسب اطلاعات بیشتر در این زمینه می توانید به فایل VB4DLL.TXT که در محل قرار گیری فایل اجرایی اصلی ویژوال بیسیک قرار دارد رجوع کنید.

به عنوان راه حلی دیگر، می توان توابع Unicode را به کمک توانایی ویژوال بیسیک در ایجاد آرایه هایی از نوع داده جدید Byte مورد استفاده قرار داد. اگر رشته های حرفی که به یک تابع خارجی ارسال می شوند یا از آن دریافت می شوند، در ویژوال بیسیک بصورت آرایه ای از نوع Byte تعریف شده باشند، ویژوال بیسیک هنگام رد و بدل کردن آنها با دنیای بیرون دیگر تبدیلی روی آنها انجام نخواهد داد.

---

[5] Type Library

مثال زیر نشان می دهد که چگونه نگارشهای ANSI (GetWindowsDirectoryA) و Unicode (GetWindowsDirectoryW) از تابع GetWindowsDirectory را تعریف کرده و فرا بخوانید.

```
Declare Function GetWindowsDirectoryA Lib "Kernel32" _
       (ByVal lpBuffer As String, ByVal nSize As Long) As Long
   Declare Function GetWindowsDirectoryW Lib "Kernel32" _
       (lpBuffer As Any, ByVal nSize As Long) As Long

   Sub mainA()
       Dim sBuf As String
       Dim cSize As Long
       Dim retval As Long

       sBuf = String(255, 0)
       cSize = 255

       retval = GetWindowsDirectoryA(sBuf, cSize)

       sBuf = Left(sBuf, retval)
       Debug.Print sBuf
   End Sub

   Sub mainW()
       Dim Buf() As Byte, sBuf As String
       Dim cSize As Long
       Dim retval As Long

       ReDim Buf(254)
       cSize = 255

       retval = GetWindowsDirectoryW(Buf(0), cSize)

       sBuf = Left(Buf, retval)
       Debug.Print sBuf
   End Sub
```

روتین mainA() نگارش ANSI تابع GetWindowsDirectory را فرا می خواند و روتین mainW() نگارش Unicode را. هر دو روتین نتایج یکسانی دارند، با اینکه روش مورد استفاده در آنها بسیار متفاوت است.

فراخوانی نگارش ANSI به همان طریقی انجام می شود که همیشه در مورد رد و بدل کردن پارامترهای رشته ای با DLL های خارجی انجام می شد. یک رشته حرفی ویژوال بیسیک تعریف می شود و با تعداد معینی کاراکتر فاصله پر می شود، و این رشته به همراه طول آن به عنوان دو پارامتر برای تابع ارسال می شوند. پس از اینکه تابع به کار خود پایان داد، کاراکترهای

فاصله اضافی از انتهای رشته حرفی بازگشتی حذف شده و مقدار نهایی استفاده می شود. عبارت ByVal در مقابل رشته حرفی در دستور تعریف تابع GetWindowsDirectoryA نشان می دهد که باید پارامتر ارسالی به DLL ، بصورت آدرسی از یک رشته حرفی باشد که با کاراکتر تهی خاتمه یافته، نه بصورت یک کپی از رشته حرفی با فرمت ویژوال بیسیک.

فراخوانی نگارش Unicode قدری متفاوت است. علاوه بر تعریف یک متغیر رشته ای به نام sBuf ، یک آرایه دینامیک از نوع Byte نیز تعریف می کنیم. پس از آن دستور ReDim طول آرایه را به میزان ۲۵۵ بایت تعریف می کند. هنگامی که GetWindowsDirectoryW فراخوانده می شود، یک آدرس که نشان دهنده اولین عنصر آرایه است به همراه اندازه آرایه برایش ارسال می شود.

تابع نیز آرایه Buf را با کدهای Unicode مربوط به کاراکترهایی که نشان دهنده دایرکتوری ویندوز هستند پر می کند و تعداد بایتهایی را که تغییر داده است بعنوان پارامتر بازگشتی ارایه می دهد. به یاد داشته باشید که از آنجا که تابع، اطلاعات را بصورت Unicode بر می گرداند هر کاراکتر دو بایت جا اشغال کرده است. برای مثال، اگر شاخه ویندوز شما C:\WINNT باشد، خانه Buf(0) از آرایه با مقدار ٦٧ پر شده، Buf(1) مقدار صفر دارد، Buf(2) معادل ۵۸ خواهد بود و Buf(3) نیز صفر است و الی آخر. در صورتیکه از کدپیجی غیر از انگلیسی استفاده شود مقادیر فوق ممکنست متفاوت باشند.

سطری از برنامه که بلافاصله پس از فراخوانی تابع GetWindowsDirectoryW قرار می گیرد، از خاصیت تبدیل آرایه بایتی به رشته حرفی استفاده می کند تا آرایه بایتی مورد نظر را به رشته حرفی معادل تبدیل نماید. هنگام تبدیل یک آرایه بایتی به یک رشته حرفی، ویژوال بیسیک فرض می کند که اطلاعات درون آرایه بصورت Unicode ذخیره شده اند. از آن به بعد متغیر sBuf حاوی رشته حرفی خواهد بود که نشان دهنده دایرکتوری ویندوز است.

مثال بالا در مورد وضعیتی است که بخواهیم یک آرایه خالی از نوع Byte را برای یک تابع ارسال کنیم. از طرف دیگر می توان یک آرایه از نوع Byte که یک رشته حرفی درون آن قرار دارد را نیز برای یک تابع خارجی ارسال کرد. برای مثال، کد زیر یک آرایه از نوع Byte را تعریف کرده، مقدار دهی نموده و برای یک تابع به نام MyPassFunction ارسال می کند.

```
Dim bArray() As Byte
bArray = "Hello" & vbNullChar
MyPassFunction bArray(0)
```

ارسال یک رشته حرفی تهی از ویژوال بیسیک برای یک تابع API

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
162622.TXT	۷ آگوست ۱۹۹۸	4 , 5 , 6	95 / 98 / NT

این بخش شرح می دهد که چگونه NULL را به صورت پارامتر برای توابع API ارسال نماییم.در لیست پارامترهای خیلی از توابع API اشاره گرهایی به رشته های حرفی وجود دارد (مانند LPSTR یا LPCSTR). اغلب مستندات این توابع اشاره کرده اند که اگر به جای رشته مورد نظر مقدار NULL ارسال شود، مثلاً فلان عملیات توسط تابع انجام می شود. در نگارشهای قدیمی تر ویژوال بیسیک می توانستید با ارسال مقدار &0 برای آن پارامتر، به نتیجه مورد نظر برسید. با این حال در نگارشهای ۴ و بالاتر ویژوال بیسیک یک ثابت مخصوص به نام vbNullString تعریف شده است که هنگامی که نیاز دارید NULL را برای توابع API ارسال کنید می توانید از این ثابت استفاده نمایید.

برای مثال، تعریف تابع FindWindow که دو پارامتر ورودی دارد را در زیر می بینید:

```
HWND FindWindow(
    LPCTSTR lpClassName,   // pointer to class name
    LPCTSTR lpWindowName   // pointer to window name
    );
```

اگر پارامتر دوم NULL باشد، FindWindow هر پنجره ای را که دارای نام کلاس خاصی باشد پیدا می کند. در برنامه C یا C++ فراخوانی این تابع می تواند شبیه این باشد:

```
hRet = FindWindow("MyWindowClass", NULL);
```

اگر همین فراخوانی را از ویژوال بیسیک انجام دهید، شبیه این خواهد بود:

```
hRet = FindWindow("MyWindowClass", vbNullString)
```

توجه کنید که استفاده از vbNullString معادل ارسال NULL به عنوان پارامتر است. ارسال یک رشته خالی ("") چنین تاثیری نخواهد داشت.

یافتن مقدار ثابت های توابع API

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
187674.TXT	۱۷ آگوست ۱۹۹۸	5 , 6	95 / 98 / NT

فایلهای راهنمای مربوط به API که همراه ویژوال بیسیک ارایه می شوند، بیشتر ثابت های API که مورد نیاز برنامه نویسان ویژوال بیسیک هستند را مشخص کرده اند. با این حال، بعضی توابع API در این فایلها مشخص نشده اند. برای یافتن ثابتهایی که توسط توابع API مورد استفاده قرار می گیرند به فایلهای Header که همراه ویژوال استودیو ۹۷ یا ویژوال C++ یا SDK پلاتفرم عرضه می شوند نیاز خواهید داشت. SDK پلاتفرم را می توانید از سایت زیر دریافت کنید:

http://www.microsoft.com/msdn/sdk/bldenv.htm

از آدرس فوق می توانید یک فایل اجرایی دریافت کنید که حاوی فایلهای Header بصورت فشرده می باشد. هنگامی که فایل را اجرا کنید، فایلهای header در شاخه \include قرار خواهند گرفت.

این فایلهای header همان هایی هستند که برای ساختن فایلهای DLL حاوی توابع API بکار رفته اند. در این بخش فرض بر این است که شما با نحوه استفاده از توابع API در ویژوال بیسیک آشنایی کافی دارید.

برای یافتن یک مقدار ثابت ابتدا باید فایل header مربوط به تابع مورد نظر خود را بیابید. برای یافتن فایل Header می توانید از تابلوی Advanced مربوط به گزینه Find Files از منوی Start سیستم عامل ویندوز استفاده نمایید. پس از یافتن فایل header این فایل را در یک ویرایشگر متنی که دارای قابلیت جستجو باشد، مانند برنامه WordPad که همراه ویندوز عرضه می شود، بازیابی کنید.

هنگامی که ثابت مورد نظر را یافتید، می توانید آنرا به همراه مقدارش در مکان مورد نظر در برنامه ویژوال بیسیک خود تعریف نمایید. مثال زیر یک ثابت در بخش Declarations از Form1 تعریف خواهد کرد.

به عنوان مثال تابع SHGetSpecialFoldersLocation مقدار PIDL مربوط به پوشه ویژه[6] مورد نظر را بر می گرداند. برای اینکه تابع بتواند PIDL را برگرداند باید نام پوشه ویژه را بصورت پارامتر برایش ارسال کنیم. در این مثال، به دنبال PIDL پوشه درایوها می گردیم که در فایل Header مربوطه یعنی SHLOBJ.H بصورت CSIDL_DRIVES تعریف شده است.

برای یافتن مقدار یک ثابت به این ترتیب عمل کنید:

۱- از منوی Start گزینه Find Files & Folders را انتخاب کنید. پنجره Find All Files ظاهر خواهد شد.

۲- در مستطیل Look In: مسیر دایرکتوری که حاوی تمام فایلهای header است (\include) را وارد کنید.

۳- روی تابلوی Advanced کلیک کنید. در فیلد Containing Text نام ثابتی را که مقدارش را می خواهید وارد کنید. در این مثال تایپ کنید : CSIDL_DRIVES

۴- روی Find Now کلیک کنید. فایل حاوی این ثابت در لیست ظاهر می شود. در مثال ما CSIDL_DRIVES در فایل SHLOBJ.H قرار دارد.

---

[6] Special Folder

۵- فایل مذکور را در یک ویرایشگر متنی مانند WordPad باز کنید. با استفاده از امکانات جستجوی ویرایشگر، نام ثابت مورد نظر را جستجو کنید. در این مثال از برنامه WordPad استفاده شده است:

- از منوی Edit گزینه Find را انتخاب کنید تا پنجره Find ظاهر شود.
- در مستطیل :Find what تایپ کنید CSIDL_DRIVES .
- روی Find Next کلیک کنید.

نتایج جستجو، مقدار CSIDL_DRIVES را معادل 0x0011 نشان می دهد. عدد 0x0011 یک عدد هگزادسیمال است که با ضوابط زبان C نمایش داده شده است. برای استفاده از این عدد در ویژوال بیسیک کافیست 0x اول آنرا با &H عوض کنید.

اگر یک عدد هگزادسیمال در محدوده 0x8000 تا 0xFFFF باشد، یک علامت & به انتهایش اضافه کنید. این کار باعث می شود ویژوال بیسیک مقدار ثابت را به یک عدد منفی تبدیل نکند. مثلا اگر مقدار ثابت برابر 0x8000 باشد برای استفاده در ویژوال بیسیک باید بصورت &H8000& بکار رود:

```
Const MY_CONSTANT = &H8000&
```

اگر به انتهای مقدار فوق علامت & را اضافه نکنید، ویژوال بیسیک مقدار فوق را بصورت &HFFFF8000 در نظر می گیرد که معادل دسیمال آن 32768- است در صورتیکه مقدار درست باید 32768 باشد.

مثال زیر نشان می دهد که چگونه می توانید از این ثابت در برنامه های ویژوال بیسیک خود استفاده کنید:

۱- یک پروژه جدید در ویژوال بیسیک ایجاد کنید. Form1 بطور پیش فرض ایجاد می شود.

۲- در بخش Declarations از Form1 سطر زیر را وارد کنید:

```
Const CSIDL_DRIVES = &H0011
```

توجه کنید که ویژوال بیسیک بطور خودکار مقدار فوق را بشکل زیر تغییر میدهد:

```
Const CSIDL_DRIVES = &H11
```

اکنون مقدار ثابت CSIDL_DRIVES به درستی در ویژوال بیسیک تعریف شده است.

کنترل پیام های ویندوز توسط AddressOf

نام فایل	تاریخ	نگارش ویژوال بیسیک	نگارش ویندوز
168795.TXT	۲۴ آگوست ۱۹۹۸	5 , 6	95 / 98 / NT

در این تکنیک که به آن کنترل پیام گفته می شود، پیام هایی که از طرف ویندوز برای یک پنجره ارسال شده اند در میانه راه در اختیار یک بخش از برنامه قرار می گیرند تا مورد بررسی و استفاده قرار گیرند. در نگارش های جدید ویژوال بیسیک می توان از اپراتور (عملگر) AddressOf در این تکنیک سود برد.

☜ توجه : هدف این بخش از کتاب صرفاً ارایه یک مثال از کاربرد روش کنترل پیام است. در بخشهای دیگری از همین کتاب مثال های دیگری از این تکنیک را مشاهده خواهید نمود.

💣 اخطار : مسئولیت هرگونه استفاده از مثال ارایه شده در این بخش به عهده خود شما خواهد بود. مایکروسافت این مثال را همینگونه که هست ارایه نموده و در مورد نحوه کار آن هیچ گونه تضمینی نمی دهد.

💣 اخطار: لازمست پیش از پاک کردن یک پنجره از حافظه، عمل کنترل پیامهای مربوط به آن پنجره را خاتمه دهید. در صورتی که این کار انجام نگیرد ممکنست باعث بروز مشکل در برنامه شده، ایجاد خطای Invalid Page Fault نموده و درنهایت باعث از دست رفتن داده ها گردد. همیشه باید پیش از پاک کردن پنجره از حافظه یا خروج از برنامه عملیات کنترل پیامهای پنجره را خاتمه دهید. این نکته خصوصاً زمان رفع اشکال از برنامه در داخل محیط ویژوال بیسیک بسیار مهم است. کلیک کردن روی دکمه توقف برنامه یا انتخاب گزینه End از منوی Run بدون خاتمه دادن به عملیات کنترل پیامها، باعث خطای Invalid Page Fault شده و ویژوال بیسیک را خواهد بست.

سیستم عامل ویندوز برای کنترل عملکرد برنامه ها پیام هایی برای پنجره های آن برنامه ها ارسال می کند. به عنوان مثال این پیام ها به پنجره اطلاع می دهند که زمان ترسیم مجدد محتویات پنجره فرارسیده است یا اینکه دکمه ای از ماوس فشار داده شده است. تمامی

اطلاعات مورد نیاز پنجره برای اینکه بتواند به درستی وظیفه خود را انجام دهد به همین طریق برایش ارسال می شود.

بـــنابراین، ساده ترین برنامه ویندوز تشکیل شده است از یک تابع (به نام WindowProc) که ایـــن پیـــام هـــا برایش ارسال می شوند. هنگامی که یک پنجره ایجاد می شود این تابع برای سیستم تعریف می شود تا سیستم بداند پیام ها را برای کجا ارسال کند.

مثال زیر تشکیل شده است از یک فرم ساده با دو Command Button. کد این برنامه طوری طراحی شده است که پیام هایی را که برای فرم فرستاده می شود دریافت کرده و محتویات آنها را در پنجره Immediate نمایش دهد.

۱- یک پروژه EXE جدید ویژوال بیسیک ایجاد کنید. Form1 بطور پیش فرض بوجود می آید.

۲- دو CommandButton به Form1 و یک ماژول به پروژه اضافه کنید.

۳- کد زیر را به بخش Declarations از Module1 اضافه نمایید:

```
Declare Function CallWindowProc Lib "user32" Alias _
  "CallWindowProcA" (ByVal lpPrevWndFunc As Long, _
  ByVal hwnd As Long, ByVal Msg As Long, _
  ByVal wParam As Long, ByVal lParam As Long) As Long

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

Public Const GWL_WNDPROC = -4
Public IsHooked As Boolean
Global lpPrevWndProc As Long
Global gHW As Long

Public Sub Hook()
    If IsHooked Then
    MsgBox "Don't hook it twice without " & _
      "unhooking, or you will be unable to unhook it."
    Else
    lpPrevWndProc = SetWindowLong(gHW, GWL_WNDPROC, _
    AddressOf WindowProc)
    IsHooked = True
    End If
End Sub

Public Sub Unhook()
    Dim temp As Long
    temp = SetWindowLong(gHW, GWL_WNDPROC,lpPrevWndProc)
    IsHooked = False
End Sub

Function WindowProc(ByVal hw As Long, ByVal uMsg As _
```

```
Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Debug.Print "Message: "; hw, uMsg, wParam, lParam
    WindowProc = CallWindowProc(lpPrevWndProc, hw, _
    uMsg, wParam, lParam)
End Function
```

۴- کد زیر را به بخش Declarations از Form1 اضافه کنید:

```
Private Sub Form_Load()
    gHW = Me.hwnd
    Command1.Caption = "Hook"
    Command2.Caption = "Unhook"
End Sub

Private Sub Command1_Click()
    Hook
End Sub

Private Sub Command2_Click()
    Unhook
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode  _
    As Integer)
    If IsHooked Then
    Cancel = 1
    MsgBox "Unhook before closing, or the IDE will crash."
    End If
End Sub
```

۵- پیش از اجرای این مثال ابتدا پروژه را ذخیره نمایید.

۶- کلید F5 را برای اجرای برنامه فشار دهید، و روی دکمه Hook کلیک کنید. ملاحظه میکنید که پنجره Immediate شروع به نمایش پیام هایی می کند که برای فرم ارسال میشوند، که البته توسط برنامه به دام انداخته شده و پس از نمایش در پنجره Immediate برای کنترل کننده پیام های خود پنجره ارسال می شوند.

دو روتین Hook و Unhook برای شروع عملیات کنترل پیام و خاتمه دادن به آن به کار میروند. در روتین اول یعنی Hook ، از تابع SetWindowLong استفاده شده است. این تابع خصوصیتی از یک پنجره مشخص را تغییر می دهد. پارامترهای مورد نیاز این تابع عبارتند از :

hwnd : شناسه پنجره ای که قرار است خصوصیتش تغییر کند.

nIndex : عملی که قرار است روی پنجره انجام گیرد.

dwNewLong : مقدار جدیدی که برای خصوصیت در نظر گرفته شده است.

در این مثال از خصوصیت hwnd فرم به عنوان شناسه پنجره مورد نظر استفاده شده است. سپس توسط ثابت GWL_WNDPROC به تابع SetWindowLong اطلاع داده می شود که قرار است آدرس روتین کنترل پیام داخلی فرم را تغییر دهید. در نهایت آدرس روتین کنترل پیام جدید در خصوصیت dwNewLong قرار داده می شود. توجه کنید که آدرس قبلی روتین کنترل پیام در متغیر lpPrevWndProc ذخیره شده است.

روتین دوم یا UnHook فقط عملی که در روتین اول انجام شده است را به حال اول در می آورد و آدرس روتین کنترل پیام اصلی را به جای خود باز می گرداند.

در این مثال تابع WindowProc روتین کنترل خطایی است که پیام های ویندوز را تحت کنترل خود می گیرد. توجه داشته باشید که با استفاده از تابع CallWindowProc و متغیر lpPrevWndProc تمامی پیام های دریافتی برای روتین کنترل خطای اصلی پنجره ارسال می شوند. به این ترتیب این امکان وجود دارد که زنجیره ای از روتین ها تمامی پیام ها را مورد پردازش قرار دهند.

مراجع :

Microsoft Windows 32 SDK

Win32 Programmer Reference
   CallWindowProc
   SetWindowLong

Microsoft Visual Basic Books Online