

ویژوال بیسیک و API ویندوز

David Shank | March 12, 2001 | MSDN

یک برنامه نویس با استفاده از توابع و اشیا ذاتی ویژوال بیسیک تنها به بخشی از سیستم عامل دسترسی و کنترل خواهد داشت. اما همین برنامه نویس با استفاده از توابع API ویندوز (Windows Application Programming Interface) قادر به کنترل ریزترین بخش های سیستم عامل خواهد بود. در این مقاله توضیحاتی درباره ی کار با توابع API از درون ویژوال بیسیک آمده است.

توجه :

فراخوانی توابع API و سایر توابع DLL می تواند برای سلامت برنامه شما خطرناک باشد. وقتی یک تابع DLL را مستقیماً از روی کد برنامه فراخوانی می کنید، در واقع یک سری از مکانیسم های مطمئنی که ویژوال بیسیک بطور معمول برای شما فراهم می کند را دور می زنید. اگر شما در تعریف یا فراخوانی یک تابع DLL اشتباه کنید باعث ایجاد یک خطا در برنامه می شوید که آن را GPF (General Protection Fault) نیز می نامند. بهترین کار برای اجرای این توابع ذخیره کردن برنامه قبل از اجرای آن است. همچنین قبل از بکارگیری یک تابع DLL باید مطمئن باشید کاملاً قواعد آن را درک کرده اید.

API ها:

یک API به بیان ساده مجموعه ای از توابع است که امکان کار با یک شیء، برنامه یا سیستم عامل را فراهم می کند. بطور معمول یک API از یک یا چند DLL تشکیل می شود که هر کدام توابع خاصی را فراهم می کنند.

فایل های DLL شامل یکسری توابع هستند که از درون هر برنامه ای در ویندوز می توان آنها را فراخوانی کرد. در هنگام اجرای چنین برنامه هایی توابع فراخوانی شده در DLL بصورت پویا به برنامه متصل میشوند. در اینکه چند برنامه بصورت همزمان یک تابع در یک DLL را فراخوانی کنند هیچ مسئله ای نیست، فایل DLL به هرحال تنها یکبار در حافظه بارگذاری می شود.

API ای که شما احتمالاً بطور مکرر راجع به آن شنیده اید، API ویندوز است. این API شامل توابعی است که سیستم عامل را تشکیل داده اند. هر برنامه تحت ویندوزی مستقیم یا غیرمستقیم با این API سروکار دارد. API ویندوز باعث می شود تا تمام برنامه های تحت ویندوز به یک شکل رفتار کنند. API های دیگری غیر از API ویندوز وجود دارند. برای مثال می توان به MAPI (Mail Application Programming Interface) اشاره کرد که مجموعه ای از فایل های DLL برای نوشتن برنامه های مرتبط با Email است.

API ها از قدیم برای برنامه نویسان C و C++ که برنامه های ویندوز را می نویسند، نوشته می شوند (البته توابع موجود در یک DLL از طریق ویژوال بیسیک هم قابل فراخوانی است). بخاطر همین فراخوانی توابع DLL از جهاتی با فراخوانی توابع در ویژوال بیسیک کمی متفاوت است و شما باید نحوه ارسال آرگومان به این توابع را یاد بگیرید.

برای استفاده از توابع API ویندوز به اسنادی احتیاج دارید که توابع قابل دسترس و نحوه تعریف و فراخوانی آنها را توضیح داده باشند. برای بدست آوردن چنین اسنادی می توانید به بخش Microsoft Platform SDK در سایت مایکروسافت رجوع کنید.

دستور Declare برای تعریف توابع DLL :

قبل فراخوانی یک تابع DLL باید اطلاعاتی راجع به مکان فایل DLL و آرگومانهای مورد نیاز آن تابع در اختیار ویژوال بیسیک قرار دهید. برای این کار دو راه وجود دارد :

- 1- قرار دادن DLL در لیست References پروژه (ارجاع به Type Library آن DLL)
- 2- استفاده از دستور Declare در یک ماژول

راه اول آسانترین راه موجود برای توابع DLL است. وقتی مرجع (reference) را مشخص می کنید می توانید تابع را چنان فراخوانی کنید که گویی جزئی داخلی در پروژه شما است. البته دو اشکال همراه این روش وجود دارد. اول آنکه ارجاع به چند Type Library بر کارایی برنامه تاثیر منفی می گذارد و دوم آنکه همه ی DLL ها Type Library ندارند. اگر می توان یک ارجاع به این نوع DLL ها هم تنظیم کرد اما نمی توان آنها را آنچنان که عضوی از برنامه باشند، فراخوانی کرد.

توجه :

هیچیک از DLL هایی که API ویندوز را تشکیل می دهند Type Library ندارند. بنابراین برای فراخوانی آنها باید از دستور Declare در بخش تعاریف ماژول (Declarations) استفاده کنید.

دستور Declare در حقیقت برای ویژوال بیسیک مشخص می کند که یک تابع DLL خاص را از کجا و چگونه می تواند فراخوانی کند. ساده ترین راه برای اضافه کردن یک عبارت Declare به کد برنامه استفاده از برنامه API Viewer است که شامل تعریف بسیاری از توابع API ویندوز، به همراه تعریف ثابت ها و نوع داده های مورد نیاز برای آن توابع است. در اینجا مثالی از نحوه تعریف تابع GetTempPath که مسیر پوشه ی Temporary ویندوز را برمی گرداند (معمولا c:\windows\temp است) آورده شده است :

```
Private Declare Function GetTempPath Lib "kernel32" _
    Alias "GetTempPathA" (ByVal nBufferLength As Long, _
    ByVal lpBuffer As String) As Long
```

کلمه Declare به ویژوال بیسیک اعلام می کند شما قصد تعریف یک تابع DLL دارید. در یک مازول استاندارد عبارت Declare می تواند بصورت Private یا Public تعریف شود بسته به اینکه شما بخواهید از آن تابع تنها در همان یک مازول یا در کل پروژه تان استفاده کنید. در یک مازول کلاس، عبارت Declare تنها بصورت Private تعریف می شود.

نامی که بعد از کلمه کلیدی Function قرار می گیرد، نام تابع است (البته نامی که شما در پروژه تان برای فراخوانی تابع استفاده می کنید). این نام می تواند کاملاً مشابه نام خود تابع API (همان نامی که در فایل DLL برای آن در نظر گرفته شده است) باشد. اگر بخواهید از نام متفاوت استفاده کنید باید نام حقیقی تابع را بعد از کلمه ی کلیدی Alias بیاورید.

در مثال فوق، نام تابع API در فایل DLL، GetTempPathA و نامی که شما می توانید تابع را در برنامه تان فراخوانی کنید GetTempPath است. توجه داشته باشید با مشخص شدن نام حقیقی تابع از هر نام مجازی که بخواهید می توانید استفاده کنید بنابراین می توانید نام هایی که API Viewer بطور پیشفرض برای توابع در نظر می گیرد را بدخواه تغییر دهید.

چرا از نام متفاوت استفاده کنیم ؟ در اینجا چند دلیل ممکن برای این کار آورده شده است :

- 1- نام بعضی از توابع DLL با کاراکتر underscore (_) شروع شده است که در ویژوال بیسیک مجاز نیست. بنابراین باید از نام دیگری استفاده کنید.
- 2- امکان نامگذاری متفاوت به شما کمک می کند تا از یک شیوه نامگذاری استاندارد و یکپارچه در برنامه تان استفاده کنید و کدهای خواناتری داشته باشید.
- 3- توابع API نسبت به حروف بزرگ و کوچک حساس هستند (case-sensitive) در حالیکه توابع ویژوال بیسیک اینگونه نیستند. بنابراین می توانید با انتخاب نام جدید case این توابع را تغییر دهید.
- 4- بعضی از توابع API آرگومان هایی دارند که چند نوع متفاوت از داده را می پذیرند. در عبارت declare این آرگومانها از نوع Any تعریف می شوند. فراخوانی توابعی که آرگومانهای Any دارند می تواند بسیار خطرناک باشد زیرا ویژوال بیسیک صحت این نوع داده ها را چک نمی کند و ممکن است داده هایی از انواع غیرمجاز به تابع ارسال شوند. برای جلوگیری از چنین وضعیتی می توان چند نسخه از یک تابع DLL، هر کدام با نام و نوع داده متفاوت تعریف کرد.
- 5- API ویندوز از تمام توابعی که آرگومان رشته ای (string) می پذیرند دو نسخه دارد: یک نسخه ANSI و یک نسخه Unicode. نسخه ANSI با پسوند A (به مثال فوق توجه کنید) و نسخه Unicode با پسوند W تمیز داده می شود. ویژوال بیسیک اگرچه در داخل از Unicode استفاده می کند اما هنگام فراخوانی یک تابع DLL بصورت خودکار تمام رشته ها را به ANSI تبدیل می کند. بنابراین شما معمولاً از نسخه ی ANSI توابع API ویندوز استفاده خواهید کرد. API ویندوز بصورت خودکار نام تمام توابعی که آگومان رشته ای دارند را تغییر می دهد و پسوندها را از آنها حذف می کند (با استفاده از کلمه کلیدی Alias)

کلمه کلیدی Lib، فایل DLL را مشخص می کند که تابع در آن قرار دارد. توجه کنید که نام DLL را باید بصورت یک رشته در تعریف Declare بگنجانید. اگر فایل DLL ای که بعد از Lib مشخص شده در سیستم کاربر پیدا نشود یک خطای زمان-اجرا با شماره 48 و مضمون "خطا در بارگذاری DLL" (Error in loading DLL) رخ خواهد داد. با این اطلاعات و مقداری کد، می توانید با این خطا مقابله کنید.

در جدول زیر DLL هایی از API ویندوز که بیشتر مورد استفاده قرار می گیرند، معرفی شده است :

DLL	مورد استفاده
Kernel.dll	توابع سطح پایین سیستم عامل، مانند توابع مربوط به مدیریت حافظه و کنترل منابع (resource handling)
User32.dll	توابع مدیریت ویندوز، مانند توابع مربوط به کنترل پیغام ها، تایمرها، منو ها و ارتباطات (communications)

GDI32.dll	(Graphic Device Interface) توابع مربوط به خروجی ها مثل رسم، نمایش و مدیریت قلم ها (fonts)
-----------	---

اکثر DLL ها بویژه آنهایی که در API ویندوز قرار دارند به زبان C و C++ نوشته شده اند بنابراین ارسال آرگومان مناسب به این DLL ها نیاز به درک آرگومانها و انواع داده در زبان های C/C++ دارد، که از جهات مختلف با ویژوال بیسیک متفاوت است.

درضمن اکثر آرگومانهای این توابع بصورت by value ارسال می شوند در حالیکه در ویژوال بیسیک آرگومانها بطور پیشفرض بصورت by reference به توابع ارسال می شوند. بنابراین ضروری است در تعریف توابعی که آرگومان را بصورت by value می پذیرند از کلمه ی کلیدی ByVal استفاده کنیم. اگر قید ByVal را فراموش کنید، ممکن است اشتباهاتی در نتایج برنامه حاصل شود. در مواردی هم ممکن است خطای زمان-اجرای با شماره ی 49 و مضمون "فراخوانی ناصحیح DLL" (Bad DLL calling convention) رخ دهد.

در ارسال آرگومان بصورت by reference، آدرس آرگومان در حافظه به تابع فراخوانی شده داده می شود و اگر تابع مقدار آرگومان را تغییر دهد، مقدار تنها نسخه از آن تغییر کرده است بنابراین آرگومان بعد از اجرای تابع مقدار جدیدی خواهد داشت.

در ارسال آرگومان بصورت by value، یک کپی از آرگومان در اختیار تابع فراخوانی شده قرار می گیرد و تابع عملیات مربوطه را روی نسخه کپی انجام می دهد بدین ترتیب مقدار اولیه آرگومان دست نخورده باقی می ماند یعنی بعد از اجرای تابع آرگومان همان مقدار قبل از اجرا را خواهد داشت.

از آنجا که در ارسال by reference امکان تغییر مقدار آرگومان وجود دارد، اگر از این نوع ارسال استفاده کنید، تابع DLL ممکن است نواحی از حافظه را که نباید بازنویسی می شدند تغییر دهد و این باعث ایجاد یک خطا یا پیدایش نتایج غیرمنتظره خواهد شد. ویندوز مقادیر زیادی دارد که نباید تغییر کنند. به عنوان مثال ویندوز به هر پنجره یک شناسه 32 بیتی واحد اختصاص می دهد که handle نامیده می شود. این شناسه ها همواره بصورت by value به تابع ارسال می شوند چرا که اگر قرار باشد توابع این شناسه ها را تغییر دهند ویندوز دیگر قادر به شناسایی آنها نخواهد بود. توجه داشته باشید که تمام رشته ها باید بصورت by reference به توابع API ویندوز ارسال شوند.

کار با ثابت ها (constants)

برخی از توابع DLL علاوه بر عبارت Declare نیازمند تعریف ثابت ها و نوع های جدیدی از داده هستند و شما باید این تعاریف اضافی را نیز در قسمت تعاریف ماژول درج کنید.

اما چگونه می توان ثابت ها و انواع جدید داده مورد نیاز برای یک تابع خاص را شناسایی کرد؟ برای این کار نیز باید به مستندات مربوط به آن تابع نگاه کنید. فایل Win32API.txt شامل تعاریف ثابت ها و انواع جدید داده می باشد. شما می توانید با استفاده از API Viewer تعاریف مورد نیاز را کپی و در کد برنامه ی خود قرار دهید. متأسفانه، در این برنامه ثابت ها و انواع جدید داده به هیچ طریقی به توابع مربوطه ارجاع داده نشده اند و شما بازهم نیاز به مستندات توابع دارید تا تشخیص دهید کدام ثابت ها و نوع ها به کدام توابع مربوطند.

توابعی وجود دارند که شما با فرستادن یک ثابت به آنها مشخص می کنید که چه اطلاعاتی از آن تابع می خواهید. به عنوان مثال، تابع GetSystemMetrics 75 ثابت می پذیرد که هرکدام مربوط به جنبه ی مختلفی از سیستم عامل است. اطلاعاتی هم که تابع برمی گرداند به همان مقدار ثابتی بستگی دارد که شما مشخص می کنید. برای فراخوانی این تابع نیاز به تعریف هر 75 ثابت وجود ندارد بلکه کافیست ثابتهای را که قصد استفاده دارید تعریف کنید.

اگرچه می توان بجای ثابت ها مقادیری را که نماینده ی آن هستند به توابع ارسال نمود، اما بهتر آن است که از خود ثابت ها استفاده کنید. مایکروسافت ثابت ماندن نام ثابت ها در نسخه های آتی ویژوال بیسیک را تضمین می کند اما هیچ تضمینی برای عدم تغییر مقدار این ثابت ها وجود ندارد.

نام ثابت های که یک تابع DLL می پذیرد معمولاً پیچیده است و نمی توان آنها را حدس زد بنابراین برای کار با توابع DLL از روی مستندات آنها با این نام ها آشنا شوید.

در مثال زیر تابع GetSystemMetrics با دستور Declare به همراه دو ثابت تعریف شده است. در این مثال دو خصیصه تعریف شده است که طول و عرض صفحه را برحسب Pixel نشان می دهند :

```
Declare Function GetSystemMetrics Lib "User32" (ByVal nIndex As Long) As Long
```

```
Const SM_CXSCREEN As Long = 0
```

```
Const SM_CYSCREEN As Long = 1
```

```
Public Property Get ScreenHeight() As Long
```

```
' Return screen height in pixels.
```

```

    ScreenHeight = GetSystemMetrics(SM_CYSCREEN)
End Property

Public Property Get ScreenWidth() As Long
    ' Return screen width in pixels.

    ScreenWidth = GetSystemMetrics(SM_CXSCREEN)
End Property

```

کار با انواع جدید داده (User-Defined Types)

یک نوع جدید داده (که برنامه نویس - همان کاربر ویژوال بیسیک- خود آنها را تعریف می کند) یک ساختار داده است که می تواند متغیرهای مرتبط با هم از انواع مختلف را کنار هم در خود نگاه دارد. در زبان های C/C++ ساختار مشابهی با این نوع داده وجود دارد. در بعضی مواقع شما یک داده ی User-Defined را به تابع DLL ارسال می کنید و تابع آنرا برای شما پر می کند و در سایر موارد عکس این روند انجام می شود.

شما می توانید داده ی User-Defined را مثل یک دراور فرض کنید که هر کشوی آن انواع مختلفی داده در خود دارد اما در کل همه ی آنها مجموعه ای از داده های مرتبط به هم هستند. در ضمن شما می توانید داده های هر کشو را مستقل از سایر کشو ها تغییر دهید.

برای تعریف یک نوع User-Defined از دستور Type ... End Type استفاده می شود. در این دستور عناصری که قرار است داده ها را ذخیره کنند با ذکر نوع آنها در مقابلشان لیست کنید. عناصر یک نوع User-Defined خود می توانند یک آرایه باشند.

قطعه کد زیر نحوه تعریف نوع جدید تابع یعنی RECT را نشان می دهد که در چند تابع API برای کنترل مستطیل های روی صفحه بکار می روند. برای مثال تابع GetWindowRect یک داده از نوع RECT می گیرد و آن را با اطلاعاتی در مورد موقعیت پنجره در صفحه پر می کند.

```

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

```

برای آنکه یک داده User-Defined را به توابع ارسال کنید باید ابتدا یک متغیر از آن نوع بسازید :

```
Private rectWindow As RECT (در قسمت تعاریف ماژول)
```

همانطور که گفته شد می توانید مستقلا به یک عنصر از داده User-Defined رجوع کنید :

```
Debug.Print rectWindow.Left
```

کار با handle ها

مورد دیگری که در مورد کار با توابع API ویندوز باید بدانید همین کار با handle ها است. handle به بیان ساده یک عدد صحیح و مثبت 32 بیتی است که ویندوز برای شناسایی یک پنجره یا شیء دیگری مانند یک font یا تصویر bitmap از آن استفاده می کند.

در سیستم عامل ویندوز، پنجره می تواند چیزهای مختلفی باشد. در حقیقت، اغلب چیزهایی که شما می توانید بر روی صفحه مشاهده کنید پنجره هستند (به اضافه ی خیلی از چیزهایی که بر روی صفحه نمی بینید). یک پنجره می تواند یک ناحیه مستطیل شکل مشخص از صفحه مانند پنجره برنامه های کاربردی باشد. یک کنترل روی فرم مانند جعبه لیست یا نوار پیمایش هم می تواند پنجره باشد، گرچه تمام کنترل ها پنجره نیستند. آیکون های روی میزکار (Desktop) و حتی خود میزکار هم پنجره هستند.

ویندوز با در نظر گرفتن تمام این اشیاء به عنوان پنجره می تواند مدیریت یکپارچه ای بر آنها داشته باشد. ویندوز به هر پنجره یک شناسه یکتا می دهد و از آن شناسه برای کار با آن پنجره استفاده می کند. بسیاری از توابع API ویندوز با handle ها سروکار دارند.

وقتی پنجره ای ساخته می شود ویندوز یک handle به آن اختصاص می دهد و وقتی آن پنجره از بین می رود handle را آزاد می کند. توجه داشته باشید که handle در طول این مدت ثابت باقی می ماند اما هیچ تضمینی وجود ندارد که وقتی همان پنجره دوباره ساخته شد همان handle قبلی را داشته

باشد. بنابراین وقتی handle يك پنجره را در يك متغير ذخيره مي كنيد بايد به ياد داشته باشيد كه آن handle فقط تا زمان بودن پنجره معتبر است نه تا زمان بودن متغير.
تابع GetActiveWindow يك نمونه از توابعي است كه handle يك پنجره را برمي گرداند (پنجره برنامه کاربردي جاري). تابع GetWindowText يك handle مي گيرد و اگر پنجره متناظر عنوان (caption) داشته باشد آن را برمي گرداند. در روال زير از اين دو تابع استفاده شده است تا عنوان پنجره ي فعال نمايش داده شود :

```
Declare Function GetActiveWindow Lib "user32" () As Long
Declare Function GetWindowText Lib "user32" _
    Alias "GetWindowTextA" (ByVal Hwnd As Long, _
    ByVal lpString As String, ByVal cch As Long) As Long

Function ActiveWindowCaption() As String
    Dim strCaption As String
    Dim lngLen As Long

    ' Create string filled with null characters.
    strCaption = String$(255, vbNullChar)
    ' Return length of string.
    lngLen = Len(strCaption)

    ' Call GetActiveWindow to return handle to active window,
    ' and pass handle to GetWindowText, along with string
    ' and its length.
    If (GetWindowText(GetActiveWindow, strCaption, _
        lngLen) > 0) Then
        ' Return value that Windows has written to string.
        ActiveWindowCaption = strCaption
    End If
End Function
```

تابع GetWindowText سه آرگومان مي پذيرد : handle يك پنجره و يك رشته مخصوص و طول اين رشته. توضيحات لازم در مورد رشته ها در ادامه ي مقاله آورده شده است.

فراخواني توابع

اگرچه فراخواني يك تابع DLL از بسياري جهات مشابه فراخواني يك تابع ويژوال بيسيك است با اين وجود تفاوت هايي وجود دارند كه توابع DLL را در نگاه نخست پيچيده به نظر مي رسانند. در ادامه به بررسي انواع آرگومانها و نحوه ي نامگذاري آنها، آرگومانها ي رشته اي، مقدار بازگشتي توابع و نحوه ي مقابله با خطاهاي خاص اين توابع خواهيم پرداخت.

انواع داده ي آرگومان ها

انواع داده اي كه در زبان C/C++ استفاده مي شوند و همچنين نكات مربوط به آنها با آنچه در ويژوال بيسيك مي دانيم، متفاوت است. در جدول زير برخي از انواع داده ي مهمي كه در توابع DLL استفاده مي شود به همراه معادل آنها در ويژوال بيسيك نمايش داده شده است :

C/C++ data type	Hungarian prefix	Description	VBA equivalent
BOOL	b	8-bit Boolean value. Zero indicates False; nonzero indicates True.	Boolean or Long
BYTE	ch	8-bit unsigned integer	Byte
HANDLE	h	32-bit unsigned integer that represents a handle to a Windows object	Long
int	n	16-bit signed integer	Integer
long	l	32-bit signed integer	Long
LP	lp	32-bit long pointer to a C/C++ structure, string, function, or other data in memory	Long
LPZSTR	lpsz	32-bit long pointer to a C-type null-terminated string	Long

اگرچه شما باید با این انواع داده آشنا باشید و پیشوندهای آنها را بشناسید، اما با استفاده از Win32API.txt دیگر نیازی به تبدیل این انواع ندارید. زیرا در عبارات Declare ای که در این فایل وجود دارد تبدیلات از قبل صورت گرفته و این عبارات بدون نیاز به دستکاری قابل استفاده در ویژوال بیسیک هستند.

رشته ها در توابع DLL

نحوه برخورد توابع DLL با رشته ها نسبت به ویژوال بیسیک متفاوت است. رشته ها همواره بصورت by reference به توابع DLL ارسال می شوند و توابع قادر به نوشتن بر روی آنها هستند. در ویژوال بیسیک شما غالباً یک رشته را به عنوان مقدار بازگشتی خود تابع برمی گردانید درحالیکه در توابع DLL یک رشته در آگومان رشته ای که به تابع ارسال شده است برگردانده می شود. مقدار بازگشتی خود این نوع توابع اغلب یک مقدار از نوع long integer است که مشخص می کند چند بایت بر روی آگومان رشته ای نوشته شده است. تابع DLL ای که یک آگومان رشته ای می پذیرد در واقع یک pointer (آدرس آن رشته در حافظه) را می گیرد. بنابراین وقتی شما در ویژوال بیسیک یک رشته به تابع DLL ارسال می کنید در واقع یک pointer به آن ارسال می کنید و تابع DLL بر روی رشته ای که آن pointer در حافظه آدرسی دهی می کند می نویسد.

برای ارسال یک رشته به توابع DLL باید چند کار بر روی آن انجام دهید. قبل از هرچیز آن رشته باید به یک کاراکتر Null ختم شود(فقط آخرین کاراکتر مهم است). این کاراکتر خاص با ثابت vbNullChar در ویژوال بیسیک شناخته می شود. نکته دوم آن است که یک تابع DLL نمی تواند طول یک رشته ساخته شده را تغییر دهد. بنابراین باید مطمئن باشید رشته ای که به تابع ارسال می کنید به اندازه کافی بزرگ باشد تا تمام مقدار بازگشتی در آن جای گیرد. در بیشتر مواقعی که یک رشته به تابع DLL ارسال می شود طول آن رشته نیز باید همراه آن ارسال شود. بدین ترتیب می تواند از نوشتن سایر برنامه ها روی خانه های حافظه ی مربوط به آن رشته جلوگیری کند. یک راه مناسب برای ارسال رشته ها به توابع DLL استفاده از تابع String\$ است. در مثال زیر یک متغیر تعریف کرده ایم و از طریق تابع String\$ آنرا با 144 کاراکتر Null پر کرده ایم :

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

اگر هنگام ارسال رشته به تابع طول آن را نمی دانید می توانید از تابع Len برای بدست آوردن آن استفاده کنید. تابع GetTempPath که آدرس پوشه ی temporary ویندوز را مشخص می کند یک نمونه از توابعی است که یک رشته را مقداردهی می کنند. این تابع دو آگومان می پذیرد : رشته و طول آن. ویندوز برای بوت شدن نیاز به وجود یک پوشه ی temporary دارد و این تابع باید همواره مسیر این پوشه را برگرداند. اگر بدیایلی چنین پوشه ای یافت نشود تابع مقدار صفر برمی گرداند. مثال زیر نحوه ی استفاده از این تابع را نشان می دهد :

```
Declare Function GetTempPath Lib "kernel32" Alias "GetTempPathA" _
    (ByVal nBufferLength As Long, ByVal lpBuffer As String) As Long
```

```
Property Get GetTempFolder() As String
    ' Returns the path to the user's Temp folder. To boot, Windows
    ' requires that a temporary folder exist, so this should always
    ' safely return a path to one. Just in case, though, check the
    ' return value of GetTempPath.
```

```
Dim strTempPath As String
Dim lngTempPath As Long
```

```
    ' Fill string with null characters.
    strTempPath = String(144, vbNullChar)
    ' Get length of string.
    lngTempPath = Len(strTempPath)
    ' Call GetTempPath, passing in string length and string.
    If (GetTempPath(lngTempPath, strTempPath) > 0) Then
        ' GetTempPath returns path into string.
        ' Truncate string at first null character.
        GetTempFolder = Left(strTempPath, _
```

```

        InStr(1, strTempPath, vbNullChar) - 1)
    Else
        GetTempFolder = ""
    End If
End Property

```

توجه داشته باشید که رشته ی ارسال شده به تابع کاملاً با کاراکتر Null پرشده است و تابع DLL مسیر را در بخش آغازی آن می نویسد و بقیه رشته را درحالیکه حاوی کاراکترهای Null است، رها می کند. در اینجا شما می توانید برای حذف کاراکترهای اضافی از تابع Left استفاده کنید.

نکته :

تنها توابعی که یک آرگومان رشته ای را مقداردهی می کنند نیاز به این اقدامات دارند. اگر تابعی رشته ای را فقط به عنوان اطلاعات مورد نیازش می پذیرد و آن را مقداردهی نمی کند کافیهست رشته را در قالب یک متغیر معمولی ویروال بیسیک به آن ارسال کنید.

ارسال داده ی User-Defined به توابع DLL

بسیاری از توابع DLL نیاز دارند تا ساختارهای از پیش تعریف شده ی ویژه ای برایشان ارسال کنید. قبل از فراخوانی این توابع باید نوع داده ی User-Defined مورد نیاز و یک متغیر از آن نوع را تعریف کرده باشید.

اینکه یک تابع به چه انواع جدید داده ای نیاز دارد را از روی عبارت Declare آن می توان تشخیص داد. متغیر آرگومان هایی که نیاز به یک نوع جدید داده دارند همواره به عنوان یک long pointer (یک مقدار 32 بیتی که ساختار نوع جدید را در حافظه مشخص می کند) تعریف می شوند. پیشوند این نوع داده "lp" است. ضمناً نوع داده ی آرگومان نام ساختار داده خواهد بود. به عنوان مثال نگاهی به تعریف توابع GetLocalTime و SetLocalTime بیاندازید :

```

Private Declare Sub GetLocalTime Lib "kernel32" _
    (lpSystem As SYSTEMTIME)
Private Declare Function SetLocalTime Lib "kernel32" _
    (lpSystem As SYSTEMTIME) As Long

```

هر دو تابع آرگومانی از نوع SYSTEMTIME می پذیرند (ساختار داده ای که شامل اطلاعاتی در مورد تاریخ و زمان است) در اینجا تعریف این نوع داده را می بینید :

```

Private Type SYSTEMTIME
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Integer
End Type

```

برای ارسال این ساختار داده به یک تابع باید یک متغیر از نوع SYSTEMTIME تعریف کنید :

```

Private sysLocalTime As SYSTEMTIME

```

هنگام فراخوانی GetLocalTime، یک متغیر از نوع SYSTEMTIME به تابع ارسال می کنیم و تابع این ساختار داده را با مقادیر عددی که مشخص کننده سال، ماه، روز، روز هفته، ساعت، دقیقه، ثانیه و میلی ثانیه محلی جاری هستند، پر می کند. برای مثال روال Property Get زیر تابع GetLocalTime را برای برگرداندن ساعت جاری فراخوانی می کند :

```

Public Property Get Hour() As Integer
    ' Retrieve current time, then return hour.

    GetLocalTime sysLocalTime
    Hour = sysLocalTime.wHour
End Property

```

هنگام فراخوانی SetLocalTime هم يك متغير از نوع SYSTEMTIME به تابع ارسال می کنیم اما قبل از يك يا چند عنصر از ساختار داده را مقداردهی می کنیم. برای مثال روال Property Get زیر ساعت سیستم زمان محلی را مقداردهی می کند.

در ابتدا GetLocalTime را فراخوانی می کند تا ساختار داده با مشخصات زمان فعلی پر شود سپس عنصر ساعت ساختار داده با مقداری که به روال ارسال شده ست می کند و در نهایت تابع SetLocalTime را با ارسال همان ساختار داده به آن فراخوانی می کند تا زمان جدید ثبت شود.

```
Public Property Let Hour(intHour As Integer)
    ' Retrieve current time so that all values will be current,
    ' then set hour portion of local time.

    GetLocalTime sysLocalTime
    sysLocalTime.wHour = intHour
    SetLocalTime sysLocalTime
End Property
```

توجه کنید که این روال چگونه فقط مقدار ساعت را تغییر می دهد.

توابع SetLocalTime و GetLocalTime کاملاً شبیه SetSystemTime و GetSystemTime هستند و تفاوت اصلی در آن است که توابع اخیر با سیستم زمانی گرینویچ تنظیم می شوند. مثلاً اگر شما در تهران زندگی کنید و ساعت محلی شما 12:00 نیمه شب باشد ساعت گرینویچ 8:30 (20:30) خواهد بود یعنی 3:30 + اختلاف زمانی.

نوع داده ی Any

برخی از توابع DLL آرگومانی دارند که می تواند انواع مختلفی از داده را بپذیرد. در تعریف این توابع این چنین آرگومان هایی با نوع داده ی Any معرفی می شوند. ویژوال بیسیک به شما اجازه می دهد هر نوع داده ای به این آرگومانها ارسال کنید این درحالیست که تابع DLL ممکن است تنها برای پذیرش دو یا سه نوع داده ی مختلف طراحی شده باشد، بنابراین ارسال داده ای با نوع اشتباه منجر به ایجاد خطا در برنامه می شود.

بطور معمول وقتی پروژه را کامپایل می کنید ویژوال بیسیک نوع داده هایی که به آرگومان ها ارسال می شوند را چک می کند و اطمینان حاصل می کند که نوع داده ی ارسالی با آنچه در تعریف آرگومان تابع آمده است، یکسان باشد. مثلاً اگر سعی کنید يك رشته (string) را به يك آرگومان Long ارسال کنید يك خطای زمان-کامپایل رخ خواهد داد. این کنترل هم برای توابع ذاتی، هم برای توابعی که برنامه نویس خود تعریف می کند و هم برای توابع DLL انجام می شود. اما توجه داشته باشید که این کنترل برای آرگومان های Any انجام نمی شود بنابراین برای مقداردهی این آرگومان ها باید بسیار مواظب باشید.

بعضی توابع DLL آرگومانی دارند که می تواند هم يك رشته و هم يك اشاره گر Null (Null Pointer) برای رشته دریافت کند. يك اشاره گر Null يك اشاره گر خاص است که به ویندوز اعلام می کند که نباید آرگومان داده شده را در نظر بگیرد. اشاره گر Null با رشته بطول صفر متفاوت است. در نسخه های قدیمی ویژوال بیسیک، برنامه نویسان مجبور بودند چنین داده هایی را با نوع داده Any تعریف کنند یا اینکه دو نسخه از تابع یکی با آرگومان String و یکی دیگر با آرگومان Long تعریف کنند. در حال حاضر ویژوال بیسیک از ثابت vbNullString برای شناسایی اشاره گر Null استفاده می کند. بنابراین کافیهست برای تعیین چنین آرگومانی از نوع String استفاده کنید و در مواقعی که نیاز به ارسال يك اشاره Null به تابع است ثابت vbNullString را بکار گیرید.

برای اطلاعات بیشتر در مورد اشاره گر Null و ویژوال بیسیک در سایت Microsoft Developer Network عبارت "vbNullString" را جستجو کنید.

مقابله با خطا

ویژوال بیسیک نسبت به خطاهای زمان اجرایی که در توابع DLL اتفاق می افتند رفتار متفاوتی دارد. هنگام رخداد اینگونه خطاها هیچ پیغامی (Message Box) نمایش داده نمی شود. وقتی خطاهای زمان-اجرا اتفاق می افتد توابع DLL مقداری برمی گردانند که خطا را مشخص می کند اما اجرای کدهای برنامه متوقف نمی شود.

بعضی از توابع API اطلاعاتی در مورد خطاهای زمان-اجرای مربوط به خود نگاهداری می کنند. اگر شما به زبان های C/C++ برنامه نویسی می کنید می توانید با تابع GetLastError از جزئیات آخرین خطای رخ داده مطلع شوید. در ویژوال بیسیک این تابع ممکن است نتایج غلطی برگرداند. برای کسب اطلاعات در مورد خطای DLL در ویژوال بیسیک می توانید از خصیصه ی LastDLLError شیء Err استفاده کنید. این خصیصه شماره ی خطای اتفاق افتاده را برمی گرداند.

برای استفاده از خصیصه LastDLLError، شما باید بدانید چه شماره ای مربوط به چه خطایی است. در این مورد نیز می توانید به سایت مایکروسافت بخش Microsoft Platform SDK رجوع کنید. مثال زیر نحوه کار با خصیصه LastDLLError را نشان می دهد. روال PrintWindowCoordinates يك handle می گیرد و تابع GetWindowRect را فراخوانی می کند. تابع GetWindowRect هم ساختار داده RECT را با مختصات مستطیلی که پنجره در آن نمایش داده می شود، پر می کند. اگر شما handle نامعتبری به تابع ارسال کنید خطایی رخ می دهد و شماره ی آن در LastDLLError ثبت می شود.

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

```
Type RECT
```

```
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
```

```
End Type
```

```
Const ERROR_INVALID_WINDOW_HANDLE As Long = 1400
```

```
Const ERROR_INVALID_WINDOW_HANDLE_DESCR As String = "Invalid window handle."
```

```
Sub PrintWindowCoordinates(hwnd As Long)
```

```
    ' Prints left, right, top, and bottom positions
    ' of a window in pixels.
```

```
    Dim rectWindow As RECT
```

```
    ' Pass in window handle and empty the data structure.
```

```
    ' If function returns 0, an error occurred.
```

```
    If GetWindowRect(hwnd, rectWindow) = 0 Then
```

```
        ' Check LastDLLError and display a dialog box if the error
        ' occurred because an invalid handle was passed.
```

```
        If Err.LastDllError = ERROR_INVALID_WINDOW_HANDLE Then
```

```
            MsgBox ERROR_INVALID_WINDOW_HANDLE_DESCR, _
                Title:="Error!"
```

```
        End If
```

```
    Else
```

```
        Debug.Print rectWindow.Bottom
```

```
        Debug.Print rectWindow.Left
```

```
        Debug.Print rectWindow.Right
```

```
        Debug.Print rectWindow.Top
```

```
    End If
```

```
End Sub
```

برای دریافت مختصات پنجره ی جاری می توانید از تابع GetActiveWindow برای بدست آوردن handle آن کمک بگیرید. کافیهست این تابع را تعریف کنید :

```
Declare Function GetActiveWindow Lib "user32" () As Long
```

و قطعه کد زیر را در پنجره Immediate وارد کنید :

```
? PrintWindowCoordinates(GetActiveWindow)
```

برای پدیدآوردن خطا می توانید تابع را با يك مقدار تصادفی فراخوانی کنید.

مستندات کامل در مورد API ویندوز را می توانید در Microsoft Platform SDK جستجو کنید.

حامد احمدی

tizfekr@yahoo.com

1383/11/19