ويژوال بيسيك و API ويندوز

David Shank | March 12,2001 | MSDN

يك برنامه نويس با استفاده از توابع و اشيا ذاتي ويژوال بيسيك تنها به بخشي از سيستم عامل دسترسي و كنترل خواهد داشت. اما همين برنامه نويس با استفاده از توابع API ويندوز (Windows Application Programming Interface) قادر به كنترل ريزترين بخش هاي سيستم عامل خواهد بود.
در اين مقاله توضيحاتي درباره ي كار با توابع API از درون ويژوال بيسيك آمده است.

توجه :
فراخواني توابع API و ساير توابع DLL مي تواند براي سلامت برنامه شما خطرناك باشد. وقتي يك تابع DLL را مستقيما از روي كد برنامه فرخواني مي كنيد، در واقع يك سري از مكانيسم هاي مطمئني كه ويژوال بيسيك بطور معمول براي شما فراهم مي كند را دور مي زنيد. اگر شما در تعريف يا فراخواني يك تابع DLL اشتباه كنيد باعث ايجاد يك خطا در برنامه مي شويد كه آن را GPF (General Protection Fault) نيز مي نامند. بهترين كار براي اجراي اين توابع ذخيره كردن برنامه قبل از اجراي آن است. همچنين قبل از بكارگيري يك تابع DLL بايد مطمئن باشيد كاملا قواعد آن را درك كرده ايد.

API ها:
يك API به بيان ساده مجموعه اي از توابع است كه امكان كار با يك شيء ، برنامه يا سيستم عامل را فراهم مي كند. بطور معمول يك API از يك يا چند DLL تشكيل مي شود كه هر كدام توابع خاصي را فراهم مي كنند.
فايل هاي DLL شامل يكسري توابع هستند كه از درون هر برنامه اي در ويندوز مي توان آنها را فراخواني كرد. در هنگام اجراي چنين برنامه هايي توابع فراخواني شده در DLL بصورت پويا به برنامه متصل ميشوند. در اينكه چند برنامه بصورت همزمان يك تابع در يك DLL را فراخواني كنند هيچ مسئله اي نيست، فايل DLL به هرحال تنها يكبار در حافظه بارگذاري مي شود.
API اي كه شما احتمالا بطور مكرر راجع به آن شنيده ايد، API ويندوز است. اين API شامل توابعي است كه سيستم عامل را تشكيل داده اند. هر برنامه تحت ويندوزي مستقيم يا غيرمستقيم با اين API سروكار دارد. API ويندوز باعث مي شود تا تمام برنامه هاي تحت ويندوز به يك شكل رفتار كنند.
API هاي ديگري غير از API ويندوز وجود دارند. براي مثال مي توان به MAPI (Mail Application Programming Interface) اشاره كرد كه مجموعه اي از فايل هاي DLL براي نوشتن برنامه هاي مرتبط با Email است.
API ها از قديم براي برنامه نويسان C و C++ كه برنامه هاي ويندوز را مي نويسند، نوشته مي شوند (البته توابع موجود در يك DLL از طريق ويژوال بيسيك هم قابل فراخواني است). بخاطر همين فراخواني توابع DLL از جهاتي با فراخواني توابع در ويژوال بيسيك كمي متفاوت است و شما بايد نحوه ارسال آرگومان به اين توابع را ياد بگيريد.
براي استفاده از توابع API ويندوز به اسنادي احتياج داريد كه توابع قابل دسترس و نحوه تعريف و فراخواني آنها را توضيح داده باشند. براي بدست آوردن چنين اسنادي مي توانيد به بخش Microsoft Platform SDK در سايت مايكروسافت رجوع كنيد.

دستور Declare براي تعريف توابع DLL :
قبل فراخواني يك تابع DLL بايد اطلاعاتي راجع به مكان فايل DLL و آرگومانها ي مورد نياز آن تابع در اختيار ويژوال بيسيك قرار دهيد. براي اين كار دو راه وجود دارد :

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

راه اول آسانترين راه موجود براي توابع DLL است. وقتي مرجع (reference) را مشخص مي كنيد مي توانيد تابع را چنان فراخواني كنيد كه گويي جزئي داخلي در پروژه شما است. البته دو اشكال همراه اين روش وجود دارد. اول آنكه ارجاع به چند Type Library بر كارايي برنامه تاثير منفي مي گذارد و دوم آنكه همه ي DLL ها Type Library ندارند. اگر مي توان يك ارجاع به اين نوع DLL ها هم تنظيم كرد اما نمي توان آنها را آنچنان كه عضوي از برنامه باشند، فراخواني كرد.

توجه :
هيچيك از DLL هايي كه API ويندوز را تشكيل مي دهند Type Library ندارند. بنابراين براي فراخواني آنها بايد از دستور Declare در بخش تعاريف ماژول (Declarations) استفاده كنيد.

دستور Declare در حقيقت براي ويژوال بيسيك مشخص مي كند كه يك تابع DLL خاص را از كجا و چگونه مي تواند فراخواني كند. ساده ترين راه براي اضافه كردن يك عبارت Declare به كد برنامه استفاده از برنامه API Viewer است كه شامل تعريف بسياري از توابع API ويندوز، به همراه تعريف ثابت ها و نوع داده هاي مورد نياز براي آن توابع است.
در اينجا مثالي از نحوه تعريف تابع GetTempPath كه مسير پوشه ي Temporary ويندوز را برمي 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
```

كلمه Declare به ويژوال بيسيك اعلام مي كند شما قصد تعريف يك تابع DLL داريد. در يك ماژول استاندارد عبارت Declare مي تواند بصورت Private يا Public تعريف شود بسته به اينكه شما بخواهيد از آن تابع تنها در همان يك ماژول يا در كل پروژه تان استفاده كنيد. در يك ماژول كلاس، عبارت Declare تنها بصورت Private تعريف مي شود.
نامي كه بعد از كلمه كليدي Function قرار مي گيرد، نام تابع است (البته نامي كه شما در پروژه تان براي فراخواني تابع استفاده مي كنيد). اين نام مي تواند كاملا مشابه نام خود تابع API (همان نامي كه در فايل DLL براي آن درنظر گرفته شده است) باشد. اگر بخواهيد از نام متفاوت استفاده كنيد بايد نام حقيقي تابع را بعد از كلمه ي كليدي Alias بياوريد.
در مثال فوق، نام تابع API در فايل DLL، GetTempPathA و نامي كه شما مي توانيد تابع را در برنامه تان فراخواني كنيد GetTempPath است. توجه داشته باشيد با مشخص شدن نام حقيقي تابع از هر نام مجازي كه بخواهيد مي توانيد استفاده كنيد بنابراين مي توانيد نام هايي كه API Viewer بطور پيشفرض براي توابع در نظر مي گيرد را بدلخواه تغيير دهيد.
چرا از نام متفاوت استفاده كنيم ؟ در اينجا چند دليل ممكن براي اين كار آورده شده است :

1- نام بعضي از توابع DLL با كاراكتر underscore (_) شروع شده است كه در ويژوال بيسيك مجاز نيست. بنابراين بايد از نام ديگري استفاده كنيد.
2- امكان نامگذاري متفاوت به شما كمك مي كند تا از يك شيوه نامگذاري استاندارد و يكپارچه در برنامه تان استفاده كنيد و كدهاي خواناتري داشته باشيد.
3- توابع API نسبت به حروف بزرگ و كوچك حساس هستند (case-sensetive) در حاليكه توابع ويژوال بيسيك اينگونه نيستند. بنابراين مي توانيد با انتخاب نام جديد case اين توابع را تغيير دهيد.
4- بعضي از توابع API آرگومان هايي دارند كه چند نوع متفاوت از داده را مي پذيرند. در عبارت declare اين آرگومانها از نوع Any تعريف مي شوند. فراخواني توابعي كه آرگومانهاي Any دارند مي تواند بسيار خطرناك باشد زيرا ويژوال بيسيك صحت اين نوع داده ها را چك نمي كند و ممكن است داده هايي از انواع غيرمجاز به تابع ارسال شوند. براي جلوگيري از چنين وضعيتي مي توان چند نسخه از يك تابع DLL، هر كدام با نام و نوع داده متفاوت تعريف كرد.
5- API ويندوز از تمام توابعي كه آرگومان رشته اي (string) مي پذيرند دو نسخه دارد: يك نسخه ANSI و يك نسخه Unicode. نسخه ANSI با پسوند A (به مثال فوق توجه كنيد)و نسخه Unicode با پسوند W تميز داده مي شود. ويژوال بسيك اگرچه در داخل از Unicode استفاده مي كند اما هنگام فراخواني يك تابع DLL بصورت خودكار تمام رشته ها را به ANSI تبديل مي كند. بنابراين شما معمولا از نسخه ي ANSI توابع API ويندوز استفاده خواهيد كرد.
API ويندوز بصورت خودكار نام تمام توابعي كه آگومان رشته اي دارند را تغيير مي دهد و پسوندها را از آنها حذف مي كند (با استفاده از كلمه كليدي Alias)

كلمه كليدي Lib، فايل DLL را مشخص مي كند كه تابع در آن قرار دارد. توجه كنيد كه نام DLL را بايد بصورت يك رشته در تعريف Declare بگنجانيد. اگر فايل DLL اي كه بعد از Lib مشخص شده در سيستم كاربر پيدا نشود يك خطاي زمان-اجرا با شماره 48 و مضمون "خطا در بارگذاري DLL" (Error in loading DLL) رخ خواهد داد. با اين اطلاعات و مقداري كد، مي توانيد با اين خطا مقابله كنيد.

در جدول زير DLL هايي از API ويندوز كه بيشتر مورد استفاده قرار مي گيرند، معرفي شده است :

DLL	مورد استفاده
Kernel.dll	توابع سطح پايين سيستم عامل، مانند توابع مربوط به مديريت حافظه و كنترل منابع (resource handling)
User32.dll	توابع مديريت ويندوز، مانند توابع مربوط به كنترل پيغام ها، تايمرها، منو ها و ارتباطات (communications)

GDI32.dll	(Graphic Device Interface) توابع مربوط به خروجي ها مثل رسم، نمايش و مديريت قلم ها (fonts)

اكثر DLL ها بويژه آنهايي كه در API ويندوز قرار دارند به زبان C و C++ نوشته شده اند بنابراين ارسال آرگومان مناسب به اين DLL ها نياز به درك آرگومانها و انواع داده در زبان هاي C/C++ دارد، كه از جهات مختلف با ويژوال بيسيك متفاوت است.

درضمن اكثر آرگومانهاي اين توابع بصورت by value ارسال مي شوند در حاليكه در ويژوال بيسيك آرگومانها بطور پيشفرض بصورت by reference به توابع ارسال مي شوند. بنابراين ضروري است در تعريف توابعي كه آرگومان را بصورت by value مي پذيرند از كلمه ي كليدي ByVal استفاده كنيم.

اگر قيد ByVal را فراموش كنيد، ممكن است اشتباهاتي در نتايج برنامه حاصل شود. در مواردي هم ممكن است خطاي زمان-اجرايي با شماره ي 49 و مضمون "فراخواني ناصحيح DLL" (Bad DLL calling convention) رخ دهد.

در ارسال آرگومان بصورت by reference، آدرس آرگومان در حافظه به تابع فراخواني شده داده مي شود و اگر تابع مقدار آرگومان را تغيير دهد، مقدار تنها نسخه از آن تغيير كرده است بنابراين آرگومان بعد از اجراي تابع مقدار جديدي خواهد داشت.

در ارسال آرگومان بصورت by value، يك كپي از آرگومان در اختيار تابع فراخواني شده قرار مي گيرد و تابع عمليات مربوطه را روي نسخه كپي انجام مي دهد بدين ترتيب مقدار اوليه آرگومان دست نخورده باقي مي ماند يعني بعد از اجراي تابع آگومان همان مقدار قبل از اجرا را خواهد داشت.

از آنجا كه در ارسال by reference امكان تغيير مقدار آرگومان وجود دارد، اگر از اين نوع ارسال استفاده كنيد، تابع DLL ممكن است نواحي از حافظه را كه نبايد بازنويسي مي شدند تغيير دهد و اين باعث ايجاد يك خطا يا پيدايش نتايج غيرمنتظره خواهد شد. ويندوز مقادير زيادي دارد كه نبايد تغيير كنند. به عنوان مثال ويندوز به هر پنجره يك شناسه 32 بيتي واحد اختصاص مي دهد كه handle ناميده مي شود. اين شناسه ها همواره بصورت by value به تابع ارسال مي شوند چرا كه اگر قرار باشد توابع اين شناسه ها را تغيير دهند ويندوز ديگر قادر به شناسايي آنها نخواهد بود.

توجه داشته باشيد كه تمام رشته ها بايد بصورت by reference به توابع API ويندوز ارسال شوند.

كار با ثابت ها (constants)

برخي از توابع DLL علاوه بر عبارت Declare نيازمند تعريف ثابت ها و نوع هايي جديدي از داده هستند و شما بايد اين تعاريف اضافي را نيز در قسمت تعاريف ماژول درج كنيد.

اما چگونه مي توان ثابت ها و انواع جديد داده مورد نياز براي يك تابع خاص را شناسايي كرد؟

براي اين كار نيز بايد به مستندات مربوط به آن تابع نگاه كنيد. فايل Win32API.txt شامل تعاريف ثابت ها و انواع جديد داده مي باشد. شما مي توانيد با استفاده از API Viewer تعاريف مورد نياز را كپي و در كد برنامه ي خود قرار دهيد. متاسفانه، در اين برنامه ثابت ها و انواع جديد داده به هيچ طريقي به توابع مربوطه ارجاع داده نشده اند و شما بازهم نياز به مستندات توابع داريد تا تشخيص دهيد كدام ثابت ها و نوع ها به كدام توابع مربوطند.

توابعي وجود دارند كه شما با فرستادن يك ثابت به آنها مشخص مي كنيد كه چه اطلاعاتي از آن تابع مي خواهيد. به عنوان مثال، تابع getSystemMetrics 75 ثابت مي پذيرد كه هركدام مربوط به جنبه ي مختلفي از سيستم عامل است. اطلاعاتي هم كه تابع برمي گرداند به همان مقدار ثابتي بستگي دارد كه شما مشخص مي كنيد. براي فراخواني اين تابع نياز به تعريف هر 75 ثابت وجود ندارد بلكه كافيست ثابتهايي را كه قصد استفاده داريد تعريف كنيد.

اگرچه مي توان بجاي ثابت ها مقاديري را كه نماينده ي آن هستند به توابع ارسال نمود، اما بهتر آن است كه از خود ثابت ها استفاده كنيد. مايكروسافت ثابت ماندن نام ثابت ها در نسخه هاي آتي ويژوال بيسيك را تضمين مي كند اما هيچ تضميني براي عدم تغيير مقدار اين ثابت ها وجود ندارد.

نام ثابت هاي كه يك تابع DLL مي پذيرد معمولا پيچيده است و نمي توان آنها را حدس زد بنابراين براي كار با توابع DLL از روي مستندات آنها با اين نام ها آشنا شويد.

در مثال زير تابع GetSystemMetrics با دستور Declare به همراه دو ثابت تعريف شده است. در اين مثال دو خصيصه تعريف شده است كه طول و عرض صفحه را برحسب Pixel نشان مي 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 جديد داده (User-Defined Types)

يك نوع جديد داده (كه برنامه نويس - همان كاربر ويژوال بيسيك- خود آنها را تعريف مي كند) يك ساختار داده است كه مي تواند متغيرهاي مرتبط با هم از انواع مختلف را كنار هم در خود نگاه دارد. در زبان هاي C/C++ ساختار مشابهي با اين نوع داده وجود دارد. در بعضي مواقع شما يك داده ي User-Defined را به تابع DLL ارسال مي كنيد و تابع آنرا براي شما پر مي كند و در ساير موارد عكس اين روند انجام مي شود.

شما مي توانيد داده ي User-Defined را مثل يك دراور فرض كنيد كه هر كشوي آن انواع مختلفي داده در خود دارد اما در كل همه ي آنها مجموعه اي از داده هاي مرتبط به هم هستند. در ضمن شما مي توانيد داده هاي هر كشو را مستقل از ساير كشو ها تغيير دهيد.

براي تعريف يك نوع User-Defined از دستور Type ... End Type استفاده مي شود. در اين دستور عناصري كه قرار است داده ها را ذخيره كنند با ذكر نوع آنها در مقابلشان ليست كنيد. عناصر يك نوع User-Defined خود مي توانند يك آرايه باشند.

قطعه كد زير نحوه تعريف نوع جديد تابع يعني RECT را نشان مي دهد كه در چند تابع API براي كنترل مستطيل هاي روي صفحه بكار مي روند. براي مثال تابع GetWindowRect يك داده از نوع RECT مي گيرد و آن را با اطلاعاتي در مورد موقعيت پنجره در صفحه پر مي كند.

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

براي آنكه يك داده User-Defined را به توابع ارسال كنيد بايد ابتدا يك متغير از آن نوع بسازيد :

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

همانطور كه گفته شد مي توانيد مستقلا به يك عنصر از داده User-Defined رجوع كنيد :

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

کار با handle ها

مورد ديگري كه در مورد كار با توابع API ويندوز بايد بدانيد همين كار با handle ها است. handle به بيان ساده يك عدد صحيح و مثبت 32 بيتي است كه ويندوز براي شناسايي يك پنجره يا شيء ديگري مانند يك font يا تصوير bitmap از آن استفاده مي كند.

در سيستم عامل ويندوز، پنجره مي تواند چيزهاي مختلفي باشد. در حقيقت، اغلب چيزهايي كه شما مي توانيد بر روي صفحه مشاهده كنيد پنجره هستند (به اضافه ي خيلي از چيزهايي كه بر روي صفحه نمي بينيد). يك پنجره مي تواند يك ناحيه مستطيل شكل مشخص از صفحه مانند پنجره برنامه هاي كاربردي باشد. يك كنترل روي فرم مانند جعبه ليست يا نوار پيمايش هم مي تواند پنجره باشد، گرچه تمام كنترل ها پنجره نيستند. آيكون هاي روي ميزكار (Desktop) و حتي خود ميزكار هم پنجره هستند.

ويندوز با درنظر گرفتن تمام اين اشياء به عنوان پنجره مي تواند مديريت يكپارچه اي بر آنها داشته باشد. ويندوز به هر پنجره يك شناسه يكتا مي دهد و از آن شناسه براي كار با آن پنجره استفاده مي كند. بسياري از توابع API ويندوز با handle ها سروكار دارند.

وقتي پنجره اي ساخته مي شود ويندوز يك handle به آن اختصاص مي دهد و وقتي آن پنجره از بين مي رود handle را آزاد مي كند. توجه داشته باشيد كه handle در طول اين مدت ثابت باقي مي ماند اما هيچ تضميني وجود ندارد كه وقتي همان پنجره دوباره ساخته شد همان handle قبلي را داشته

باشد. بنابراين وقتي 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 كاربردي 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 بايد با اين انواع داده آشنا باشيد و پيشوندهاي آنها را بشناسيد، اما با استفاده از Win32API.txt ديگر نيازي به تبديل اين انواع نداريد. زيرا در عبارات Declare اي كه در اين فايل وجود دارد تبديلات از قبل صورت گرفته و اين عبارات بدون نياز به دستكاري قابل استفاده در ويژوال بيسيك هستند.

رشته ها در توبع DLL

نحوه برخورد توابع DLL با رشته ها نسبت به ويژوال بيسيك متفاوت است. رشته ها همواره بصورت by reference به توابع DLL ارسال مي شوند و توابع قادر به نوشتن بر روي آنها هستند.

در ويژوال بيسيك شما غالبا يك رشته را به عنوان مقدار بازگشتي خود تابع برمي گردانيد درحاليكه در توابع DLL يك رشته در آگومان رشته اي كه به تابع ارسال شده است برگردانده مي شود. مقدار بازگشتي خود اين نوع توابع اغلب يك مقدار از نوع long integer است كه مشخص مي كند چند بايت بر روي آرگومان رشته اي نوشته شده است.

تابع DLL اي كه يك آرگومان رشته اي مي پذيرد در واقع يك pointer (آدرس آن رشته در حافظه) را مي گيرد. بنابراين وقتي شما در ويژوال بيسيك يك رشته به تابع DLL ارسال مي كنيد در واقع يك pointer به آن ارسال مي كنيد و تابع DLL بر روي رشته اي كه آن pointer در حافظه آدرسي دهي مي كند مي نويسد.

براي ارسال يك رشته به توابع DLL بايد چند كار بر روي آن انجام دهيد. قبل از هرچيز آن رشته بايد به يك كاراكتر Null ختم شود(فقط آخرين كاراكتر مهم است). اين كاراكتر خاص با ثابت vbnullChar در ويژوال بيسيك شناخته مي شود. نكته دوم آن است كه يك تابع DLL نمي تواند طول يك رشته ساخته شده را تغيير دهد. بنابراين بايد مطمئن باشيد رشته اي كه به تابع ارسال مي كنيد به اندازه كافي بزرگ باشد تا تمام مقدار بازگشتي در آن جاي گيرد.در بيشتر مواقعي كه يك رشته به تابع DLL ارسال مي شود طول آن رشته نيز بايد همراه آن ارسال شود. بدين ترتيب مي تواند از نوشتن ساير برنامه ها روي خانه هاي حافظه ي مربوط به آن رشته جلوگيري كند.

يك راه مناسب براي ارسال رشته ها به توابع DLL استفاده از تابع $String است. در مثال زير يك متغير تعريف كرده ايم و از طريق تابع $String آنرا با 144 كاراكتر Null پر كرده ايم :

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

اگر هنگام ارسال رشته به تابع طول آن را نمي دانيد مي توانيد از تابع Len براي بدست آوردن آن استفاده كنيد.

تابع GetTempPath كه آدرس پوشه ي temporary ويندوز را مشخص مي كند يك نمونه از توابعي است كه يك رشته را مقداردهي مي كنند. اين تابع دو آرگومان مي پذيرد : رشته و طول آن.

ويندوز براي بوت شدن نياز به وجود يك پوشه ي temporary دارد و اين تابع بايد همواره مسير اين پوشه را برگرداند. اگر بدلايلي چنين پوشه اي يافت نشود تابع مقدار صفر برمي گرداند.

مثال زير نحوه ي استفاده از اين تابع را نشان مي 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 باشيد كه رشته ي ارسال شده به تابع كاملا با كاراكتر Null پرشده است و تابع DLL مسير را در بخش آغازي آن مي نويسد و بقيه رشته را درحاليكه حاوي كاراكترهاي Null است، رها مي كند. در اينجا شما مي توانيد براي حذف كاراكترهاي اضافي از تابع Left استفاده كنيد.

نكته :

تنها توابعي كه يك آرگومان رشته اي را مقداردهي مي كنند نياز به اين اقدامات دارند. اگر تابعي رشته اي را فقط به عنوان اطلاعات مورد نيازش مي پذيرد و آن را مقداردهي نمي كند كافيست رشته را در قالب يك متغير معمولي ويژوال بيسيك به آن ارسال كنيد.

ارسال داده ي User-Defined به توابع DLL

بسياري از توابع DLL نياز دارند تا ساختارهاي از پيش تعريف شده ي ويژه اي برايشان ارسال كنيد. قبل از فراخواني اين توابع بايد نوع داده ي User-Defined مورد نياز و يك متغير از آن نوع را تعريف كرده باشيد.

اينكه يك تابع به چه انواع جديد داده اي نياز دارد را از روي عبارت Declare آن مي توان تشخيص داد. متغير آرگومان هايي كه نياز به يك نوع جديد داده دارند همواره به عنوان يك long pointer (يك مقدار 32 بيتي كه ساختار نوع جديد را در حافظه مشخص مي كند) تعريف مي شوند. پيشوند اين نوع داده "lp" است. ضمنا نوع داده ي آرگومان نام ساختار داده خواهد بود.

به عنوان مثال نگاهي به تعريف توابع GetLocalTime و SetLocalTime بياندازيد :

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

هر دو تابع آرگوماني از نوع SYSTEMTIME مي پذيرند (ساختار داده اي كه شامل اطلاعاتي در مورد تاريخ و زمان است) در اينجا تعريف اين نوع داده را مي بينيد :

```
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
```

براي ارسال اين ساختار داده به يك تابع بايد يك متغير از نوع SYSTEMTIME تعريف كنيد :

```
Private sysLocalTime As SYSTEMTIME
```

هنگام فراخواني GetLocalTime، يك متغير از نوع SYSTEMTIME به تابع ارسال مي كنيم و تابع اين ساختار داده را با مقادير عددي كه مشخص كننده سال، ماه، روز، روز هفته، ساعت، دقيقه، ثانيه و ميلي ثانيه محلي جاري هستند، پر مي كند. براي مثال روال Proprty Get زير تابع GetLocalTime را براي برگرداندن ساعت جاري فراخواني مي كند :

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

   GetLocalTime sysLocalTime
   Hour = sysLocalTime.wHour
End Property
```

هنگام فراخواني SetLocalTime هم يك متغير از نوع SYSTEMTIME به تابع ارسال مي كنيم اما قبل از يك يا چند عنصر از ساختار داده را مقداردهي مي كنيم. براي مثال روال Propety Get زير ساعت *سيستم زمان محلي* را مقداردهي مي كند.
در ابتدا GetLocalTime را فراخواني مي كند تا ساختار داده با مشخصات زمان فعلي پر شود سپس عنصر ساعت ساختار داده با مقداري كه به روال ارسال شده ست مي كند و در نهايت تابع SetLocalTime را با ارسال همان ساختار داده به آن فراخواني مي كند تا زمان جديد 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 كنيد كه اين روال چگونه فقط مقدار ساعت را تغيير مي دهد.
توابع SetLocalTime و GetLocalTime كاملا شبيه SetSystemTime و GetSystemTime هستند و تفاوت اصلي در آن است كه توابع اخير با سيستم زماني گرينويچ تنظيم مي شوند. مثلا اگر شما در تهران زندگي كنيد و ساعت محلي شما 12:00 نيمه شب باشد ساعت گرينويچ 8:30 (20:30) خواهد بود يعني 3:30 + اختلاف زماني.

نوع داده ي Any

برخي از توابع DLL آرگوماني دارند كه مي تواند انواع مختلفي از داده را بپذيرد. در تعريف اين توابع اين چنين آرگومان هايي با نوع داده ي Any معرفي مي شوند. ويژوال بيسيك به شما اجازه مي دهد هر نوع داده اي به اين آرگومانها ارسال كنيد اين درحاليست كه تابع DLL ممكن است تنها براي پذيرش دو يا سه نوع داده ي مختلف طراحي شده باشد، بنابراين ارسال داده اي با نوع اشتباه منجر به ايجاد خطا در برنامه مي شود.
بطور معمول وقتي پروژه را كامپايل مي كنيد ويژوال بيسيك نوع داده هايي كه به آرگومان ها ارسال مي شوند را چك مي كند و اطمينان حاصل مي كند كه نوع داده ي ارسالي با آنچه در تعريف آرگومان تابع آمده است، يكسان باشد. مثلا اگر سعي كنيد يك رشته (string) را به يك آرگومان Long ارسال كنيد يك خطاي زمان-كامپايل رخ خواهد داد. اين كنترل هم براي توابع ذاتي، هم براي توابعي كه برنامه نويس خود تعريف مي كند و هم براي توابع DLL انجام مي شود. اما توجه داشته باشيد كه اين كنترل براي آرگومان هاي Any انجام نمي شود بنابراين براي مقداردهي اين آرگومان ها بايد بسيار مواظب باشيد.
بعضي توابع DLL آرگوماني دارند كه مي تواند هم يك رشته و هم يك اشاره گر Null (Null Pointer) براي رشته دريافت كند. يك اشاره گر Null يك اشاره گر خاص است كه به ويندوز اعلام مي كند كه نبايد آرگومان داده شده را درنظر بگيرد. اشاره گر Null با رشته بطول صفر متفاوت است. در نسخه هاي قديمي ويژوال بيسيك، برنامه نويسان مجبور بودند چنين داده هايي را با نوع داده Any تعريف كنند يا اينكه دو نسخه از تابع يكي با آرگومان String و يكي ديگر با آرگومان Long تعريف كنند. در حال حاضر ويژوال بيسيك از ثابت vbNullString براي شناسايي اشاره گر Null استفاده مي كند. بنابراين كافيست براي تعيين چنين آرگوماني از نوع String استفاده كنيد و در مواقعي كه نياز به ارسال يك اشاره Null به تابع است ثابت vbNullString را بكار گيريد.
براي اطلاعات بيشتر در مورد اشاره گر Null ويژوال بيسيك در سايتMicrosoft Developer Network عبارت "vbNullString" را جستجو كنيد.

مقابله با خطا

ويژوال بيسيك نسبت به خطاهاي زمان اجرايي كه در توابع DLL اتفاق مي افتند رفتار متفاوتي دارد. هنگام رخداد اينگونه خطاها هيچ پيغامي (Message Box) نمايش داده نمي شود. وقتي خطاهاي زمان-اجرا اتفاق مي افتد توابع DLL مقداري برمي گردانند كه خطا را مشخص مي كند اما اجراي كدهاي برنامه متوقف نمي شود.
بعضي از توابع API اطلاعاتي در مورد خطاهاي زمان-اجراي مربوط به خود نگاهداري مي كنند. اگر شما به زبان هاي C/C++ برنامه نويسي مي كنيد مي توانيد با تابع GetLastError از جزئيات آخرين خطاي رخ داده مطلع شويد. در ويژوال بيسيك اين تابع ممكن است نتايج غلطي برگرداند. براي كسب اطلاعات در مورد خطاي DLL در ويژوال بيسيك مي توانيد از خصيصه ي LastDLLError شيء Err استفاده كنيد. اين خصيصه شماره ي خطاي اتفاق افتاده را برمي گرداند.

براي استفاده از خصيصه LastDLLError، شما بايد بدانيد چه شماره اي مربوط به چه خطايي است. در اين مورد نيز مي توانيد به سايت مايكروسافت بخش Microsoft Platform SDK رجوع كنيد.
مثال زير نحوه كار با خصيصه ي LastDLLError را نشان مي دهد. روال PrintWindowCoordinates يك handle مي گيرد و تابع GetWindowRect را فراخواني مي كند. تابع GetWindoeRect هم ساختار داده ي RECT را با مختصات مستطيلي كه پنجره در آن نمايش داده مي شود، پر مي كند. اگر شما handle نامعتبري به تابع ارسال كنيد خطايي رخ مي دهد و شماره ي آن در LastDLLError ثبت مي 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
```

براي دريافت مختصات پنجره ي جاري مي توانيد از تابع GetActiveWindow براي بدست آوردن handle آن كمك بگيريد. كافيست اين تابع را تعريف كنيد :

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

و قطعه كد زير را در پنجره Immediate وارد كنيد :

```
? PrintWindowCoordinates(GetActiveWindow)
```

براي پديدآوردن خطا مي توانيد تابع را با يك مقدار تصادفي فراخواني كنيد.
مستندات كامل در مورد API ويندوز را مي توانيد در Microsoft Platform SDK جستجو كنيد.

حامد احمدي
tizfekr@yahoo.com
1383/11/19