استفاده از توابع API در Net.

یکی از سوالاتی که برای بسیاری از مبتدیان در دنیای برنامه نویسی NET. مطرح می باشد، این است که آیا در NET. می توان به توابع API ویندوز دسترسی پیدا کرد یا خیر ؟! بسیاری گمان می کنند که با آمدن NET.، توابع API ویندوز بلااستفاده شده اند و دیگر به درد نمی خورند! اما اینطور نیست. با وجود اینکه NET Class Library. بسیاری از کارها را برای برنامه نویسان، ساده و Object-Oriented ساخته، اما هنوز در مواردی لازم است که سراغ API های ویندوز برویم و اعمال خاصی را انجام دهیم که در کلاس های NET. پیاده سازی نشده اند. در این مقاله نحوه ارتباط با API های ویندوز را یاد خواهیم گرفت.

Platform Invoke **چیست ؟**

Platform Invoke یکی از قابلیت های NET. است که به شما این امکان را می دهد که API های ویندوز را در Managed Code صدا بزنید. برای استفاده از API های ویندوز، ابتدا باید نام تابع مورد نظر و نام DLL ای که این تابع در آن تعریف شده است را پیدا کنید. در MSDN کلیه API ها بر اساس موضوع طبقه بندی شده اند که مي توانید با فیلتر کردن مطالب Platform SDK آن ها را راحت تر بیابید. در پایین صفحه مربوط به توضیحات هر تابع، نام DLL ای که آن تابع در آنجا تعریف شده آمده است. معمولاً ۳ DLL کلی هستند که توابع ویندوز در آن ها تعریف شده اند:

۱. User32.DLL: شامل توابع لازم برای ارتباط با منوها، Timer ها و ...
۲. GDI32.DLL: شامل توابع گرافیکی و فونت
۳. Kernel32.DLL: شامل توابع Low-Level سیستم عامل

پس از اینکه نام تابع و DLL مورد نظر را پیدا کردید، باید با استفاده از DllImport آن را به برنامه تان متصل کنید. در این مرحله باید Data Type های تابع مورد نظر را به Data Type هاي معادل آنها در Managed Code تبدیل کنید. اگر تا به حال برنامه نویسی ویندوز با C++ نکرده باشید، تشخیص اینکه کدام Data Type در NET. معادل Data Type مورد نظر در Windows است برای شما مشکل خواهد بود. با این حال، لیستی از پرکاربردترین Data Type های ویندوز و معادل آن ها در NET. را در جدول زیر آورده ام. در صورتی

که می خواهید جزئیات بیشتر درباره نحوه تبدیل Data Type های ویندوز به NET. بدانید، پیشنهاد می کنم مبحث Marshaling Data < Platform Invoke را در MSDN مطالعه نمایید. جهت پیدا کردن این مطلب، در قسمت Index بنویسید Platform Invoke. سپس لیستی از آیتم هایی که به این مبحث مربوط می شوند در بخش Index ظاهر می شوند. در این لیست، گزینه Marshaling Data را کلیک کرده و مطالعه نمایید. توجه داشته باشید که فیلتر باید بر روی گزینه (filter no) باشد. در غیر اینصورت مبحث Platform Invoke را در لیست Index نخواهید یافت.

در جدول زیر، لیستی از Data Type های ویندوز به همراه معادل آن ها در NET. را می بینید. به ستون اول دقت کنید. Data Type هایی که در این ستون می بینید، به کرات در API های ویندوز استفاده شده اند. معادل هر Data Type در ویندوز، در ستون دوم (Managed class name) آمده است.

Unmanaged type in Wtypes.h	Managed class name
HANDLE	System.IntPtr
BYTE	System.Byte
SHORT	System.Int16
WORD	System.UInt16
INT	System.Int32
UINT	System.UInt32
LONG	System.Int32
BOOL	System.Int32
DWORD	System.UInt32
ULONG	System.UInt32
CHAR	System.Char
LPSTR	System.String or System.StringBuilder

LPCSTR	System.String or System.StringBuilder
LPWSTR	System.String or System.StringBuilder
LPCWSTR	System.String or System.StringBuilder
FLOAT	System.Single
DOUBLE	System.Double

یک مثال

در اینجا می خواهیم از تابع GetComputerName که در ویندوز به صورت زیر تعریف شده است در برنامه مان استفاده کنیم:

```
int GetComputerName(LPSTR buffer, LPDWORD size);
```

به خاطر داشته باشید که LP در ویندوز به معنای Pointer Long است و در مواردی به کار می رود که قرار باشد یک متغیر با آدرس به API ارسال شود. در #C، در صورتی که از انواع Value Type (مثل int، char، float و ...) استفاده می کنید، لازم است قبل از نام متغیر، کلمه کلیدی ref (به معنی reference) را اضافه کنید. در صورتی که از انواع Reference Type (مثل کلاس ها) استفاده می کنید، نیازی به استفاده از این کلمه کلیدی نیست.

بنابراین، با توجه به جدول فوق، می توان تابع GetComputerName() را به صورت زیر در .NET. به برنامه متصل کرد:

```
public class Win32API
{
     [DllImport("kernel32.dll", CharSet = CharSet.Auto)]
     public static extern int GetComputerName(StringBuilder buffer, ref
uint size);
```

```
}
```

DllImport یک Attribute است که به CLR می گوید که این تابع در کجا تعریف شده است. آرگومان دوم این Attribute ، یعنی CharSet = CharSet.Auto برای فراخوانی توابعی که از ۲ فرمت ANSI و Unicode استفاده می کنند لازم است. بسیاری از توابع ویندوز (مانند GetComputerName) در ۲ فرمت می آیند: GetComputerNameA و GetComputerNameW. در این موارد، لازم است که تشخیص اینکه از کدام نسخه استفاده شود را به CLR واگذار کنید. عبارت CharSet.Auto این کار را به طور اتوماتیک انجام می دهد.

در اینجا، ما یک کلاس به نام Win32API تعریف کرده ایم و توابع API ای که می خواهیم از آنها استفاده کنیم را در این کلاس قرار خواهیم داد. شما هم می توانید API هایی که زیاد با آنها کار می کنید را در یک یا چند کلاس از نظر موضوعی دسته بندی کنید. به چنین کلاسی Wrapper گفته می شود. این تکنیک در نگهداری برنامه ها و استفاده مکرر از کد تاثیر بسزایی دارد.

یک نکته مهم: در هنگام Declare توابع API، توجه داشته باشید که این توابع را باید همواره از نوع static و extern تعریف کنید تا CLR بتواند آن ها را در DLL های مربوطه پیدا کند.

اما نحوه استفاده از این تابع! فرض کنید که می خواهیم اسم کامپیوتر را با استفاده از تابع GetComputerName() گرفته و در یک رشته به نام ComputerName قرار دهیم. برای این منظور، می نویسیم:

```
StringBuilder buffer = new StringBuilder(128);
uint size = (uint)buffer.Capacity;
Win32API.GetComputerName(buffer, ref size);
string ComputerName = buffer.ToString();
```

البته، فراخوانی API های ویندوز پیچیدگی ها و مشکلات زیادی دارد که توضیح آن ها از حوصله این مقاله خارج است. در این مقاله فقط می خواستیم کلیاتی درباره نحوه فرخوانی API ها در NET. را یاد بگیریم. جهت مطالعه جزئیات بیشتر درباره این مبحث، مطلب Platform Invoke را در MSDN مطالعه نمایید.