C++/CLI y C# VII: GetLastError en escenarios de interop

Introducción

Hay un tema que ha aparecido indirectamente en algunos mensajes del grupo de C# relativo al tema de Interop mediante atributos (que es el único que puede hacer C# y VB.NET), y que desde que estoy leyendo dicho grupo nadie ha advertido o ha tenido algún problema con él, o al menos yo no me he dado cuenta de ello.

Cuando uno hace una llamada al API de Win32, muchas funciones devuelven un estado de error a través de un *handle* inválido o simplemente devolviendo *FALSE*. Es una consecuencia de los tiempos en que ni el compilador ni Windows soportaban ningún modelo de excepción. Era tarea del programador comprobar dicho valor de retorno y luego realizar una llamada a *GetLastError(),* que devolvía un número y que se podía convertir a una cadena más o menos inteligible mediante una nueva llamada a otra función: *FormatMessage*.

Si ahora volvemos al entorno manejado, cuando queramos realizar una llamada a través de interop a cualquier función de Win32 que utiliza ese formato para retornar un error, nos encontramos con la duda de qué ocurre entre las dos llamadas, es decir, nuestra llamada al método y una segunda para obtener el error.

El Interop por atributos funciona de la siguiente forma: cuando nosotros hacemos una llamada a través de este sistema, el compilador inserta una cosa llamada *thunk*, que consiste en una llamada a ejecutar cierto código especial. Entre muchas otras cosas, este código se encarga de traer la DLL de Win32 a nuestro espacio de direcciones (es un decir, porque realmente no trae nada, sino que prepara el contexto para realizar la llamada), realizar la conversión de los parámetros en parámetros compatibles para Win32, ejecutar dicha función y volver a convertir todo el tema en valores que el .NET sea capaz de entender. Lento, muy lento.

Cuando volvamos a llamar a otra función, la operación se repite, por lo que una vez que se ha realizado la llamada a la función que queríamos, cuando hagamos la nueva llamada a *GetLastError()* repetiremos todo lo de arriba, de forma que hemos perdido el valor, ya que estamos en un nuevo contexto que nada tiene que ver con el anterior.

Si seguimos las reglas tal y como las hemos planteado, el tema tendría un fallo garrafal en cuanto a funcionalidad hacia el API Win32… Pero lo cierto es que los chicos de MS no son tontos, y han solucionado el problema de forma bastante elegante, aunque ineficaz desde un punto de vista del rendimiento, pero creo que excepto permitiendo el Interop al estilo en que se permite en C++/CLI, es la única forma posible de hacerlo.

Cómo funciona por defecto

Cuando hacemos una llamada al API Win32, el sistema de Interop hace por nosotros la llamada a *GetLastError()* en cada llamada, de forma que dicho resultado se guarda en el contexto. Y podemos accederlo mediante el método *Marshal.GetLastWin32Error().*

Es decir, cada vez que hagamos una llamada mediante Interop a una función del API de Win32, tendremos disponible el valor devuelto por el *GetLastError()* de Win32, ya que dicha llamada se ha hecho automáticamente.

Activarlo y desactivarlo

Este funcionamiento viene activado por defecto, y lo cierto es que añade más sobrecarga a un sistema ya sobrecargado de por sí ya que, ante cada llamada a una función de Windows, el sistema hace dos, además de guardar un valor en el TLS (Thread Local Storage) del contexto de ejecución en el que se haya realizado. Y nosotros tendremos que realizar también una nueva consulta a *Marshal.GetLastWin32Error()*, y si queremos enseñar un mensaje bonito, a *FormatMessage()* de nuevo, que no es una función sencilla de utilizar y que requiere también un buen trabajo de atributos y de interop.

Una llamada a un método de USER32.DLL podría tener esta firma:

```
[System.Runtime.InteropServices.DllImport("user32.dll")]
extern static void SampleMethod();
```

De esta forma, estamos indicando el funcionamiento por defecto, es decir, poder obtener el valor del error de Win32.

Pero si especificamos esta forma:

```
[System.Runtime.InteropServices.DllImport("user32.dll")]
SetLastError=false]
extern static void SampleMethod();
```

Estamos indicando al sistema de Interop que no realice la llamada a *GetLastError()*, de forma que optimizamos en cierta medida los accesos a Win32 siempre y cuando no nos interese determinar si hubo error o no.

Un ejemplo podría ser una llamada a *MessageBeep()*. Si falla poco podemos hacer, ya que el usuario no oirá nada y punto.