How to: Implement callback functions

Article • 08/16/2023

The following procedure and example demonstrate how a managed application, using platform invoke, can print the handle value for each window on the local computer. Specifically, the procedure and example use the **EnumWindows** function to step through the list of windows and a managed callback function (named CallBack) to print the value of the window handle.

To implement a callback function

1. Look at the signature for the **EnumWindows** function before going further with the implementation. **EnumWindows** has the following signature:

   ```
   C++

   BOOL EnumWindows(WNDENUMPROC lpEnumFunc, LPARAM lParam)
   ```

   One clue that this function requires a callback is the presence of the **lpEnumFunc** argument. It is common to see the **lp** (long pointer) prefix combined with the **Func** suffix in the name of arguments that take a pointer to a callback function. For documentation about Win32 functions, see the Microsoft Platform SDK.

2. Create the managed callback function. The example declares a delegate type, called `CallBack`, which takes two arguments (**hwnd** and **lparam**). The first argument is a handle to the window; the second argument is application-defined. In this release, both arguments must be integers.

   Callback functions generally return nonzero values to indicate success and zero to indicate failure. This example explicitly sets the return value to **true** to continue the enumeration.

3. Create a delegate and pass it as an argument to the **EnumWindows** function. Platform invoke converts the delegate to a familiar callback format automatically.

4. Ensure that the garbage collector does not reclaim the delegate before the callback function completes its work. When you pass a delegate as a parameter, or pass a delegate contained as a field in a structure, it remains uncollected for the duration of the call. So, as is the case in the following enumeration example, the callback function completes its work before the call returns and requires no additional action by the managed caller.

If, however, the callback function can be invoked after the call returns, the managed caller must take steps to ensure that the delegate remains uncollected until the callback function finishes. For an example, see the GCHandle sample.

Example

C#

```
using System;
using System.Runtime.InteropServices;

public delegate bool CallBack(int hwnd, int lParam);

public class EnumReportApp
{
    [DllImport("user32")]
    public static extern int EnumWindows(CallBack x, int y);

    public static void Main()
    {
        CallBack myCallBack = new CallBack(EnumReportApp.Report);
        EnumWindows(myCallBack, 0);
    }

    public static bool Report(int hwnd, int lParam)
    {
        Console.Write("Window handle is ");
        Console.WriteLine(hwnd);
        return true;
    }
}
```

See also

- Callback Functions
- Calling a DLL Function

more information, see our contributor guide.

Provide product feedback