Common gotchas when writing your own p/invoke

devblogs.microsoft.com/oldnewthing/20090813-00

August 13, 2009

Raymond Chen

If you're looking to get into some p/invoke action, you'd be well-served to check out the pinvoke wiki to see if somebody else has done it too. If what you need isn't there, you may end up forced to write your own, and here are some gotchas I've seen people run into:

- C++ `bool` and Win32 `BOOLEAN` are not the same as C# `bool` (aka `System.Boolean`). In Win32, `BOOL` is a 4-byte type, and `BOOLEAN` is a 1-byte type. [*See also MadQ's remarks about `VARIANT_BOOL`.*] Meanwhile, C++ `bool` is not standardized by Win32, so the size will vary based on your compiler, but most compilers use a 1-byte value. And then C# is even weirder: The `bool` is a 1-byte type, but it marshals as a 4-byte type by default.
- Win32 `char` is not the same as C# `char` (aka `System.Char`). In C#, `char` is a Unicode character (two bytes), whereas in C/C++ under Win32 it is an ANSI character (one byte).
- Win32 `long` is not the same as C# `long` (aka `System.Int64`). In C#, `long` is 64-bit value, whereas in C/C++ under Win32 it is a 32-bit value.
- If memory is allocated and freed across the interop boundary, make sure both sides are using the same allocator. It is my understanding that the CLR uses `CoTaskMemAlloc`/`CoTaskMemFree` by default. If your Win32 function doesn't use `CoTaskMemAlloc`, you'll have to teach the CLR which allocator you really want.
- When laying out structures, you have to watch out for alignment.

That last one is particularly gnarly on 64-bit systems, where alignment requirements are less forgiving than on x86. The structure declarations on pinvoke.net tend to ignore 64-bit issues. For example, the declaration of the `INPUT` structure (as of this writing—it's a wiki so it's probably changed by the time you read this) reads as follows:

```
[StructLayout(LayoutKind.Explicit)]struct INPUT {
  [FieldOffset(0)] int type;
  [FieldOffset(4)] MOUSEINPUT mi;
  [FieldOffset(4)] KEYBDINPUT ki;
  [FieldOffset(4)] HARDWAREINPUT hi;
}
```

This structure layout is correct for 32-bit Windows, but it's incorrect for 64-bit Windows.

Let's take a look at that `MOUSEINPUT` structure, for starters.

```
typedef struct tagMOUSEINPUT {
    LONG    dx;
    LONG    dy;
    DWORD   mouseData;
    DWORD   dwFlags;
    DWORD   time;
    ULONG_PTR dwExtraInfo;
} MOUSEINPUT, *PMOUSEINPUT, FAR* LPMOUSEINPUT;
```

In 64-bit Windows, the `LONG` and `DWORD` members are four bytes, but the `dwExtraInfo` is a `ULONG_PTR`, which is eight bytes on a 64-bit machine. Since Windows assumes /Zp8 packing, the `dwExtraInfo` must be aligned on an 8-byte boundary, which forces four bytes of padding to be inserted after the `time` to get the `dwExtraInfo` to align properly. And in order for all this to work, the `MOUSEINPUT` structure itself must be 8-byte aligned.

Now let's look at that `INPUT` structure again. Since the `MOUSEINPUT` comes after the `type`, there also needs to be padding between the `type` and the `MOUSEINPUT` to get the `MOUSEINPUT` back to an 8-byte boundary. In other words, the offset of `mi` in the `INPUT` structure is 8 on 64-bit Windows, not 4.

Here's how I would've written it:

```
// This generates the anonymous union
[StructLayout(LayoutKind.Explicit)] struct INPUT_UNION {
  [FieldOffset(0)] MOUSEINPUT mi;
  [FieldOffset(0)] KEYBDINPUT ki;
  [FieldOffset(0)] HARDWAREINPUT hi;
};
[StructLayout(LayoutKind.Sequential)] struct INPUT {
  int type;
  INPUT_UNION u;
}
```

I introduce a helper structure to represent the anonymous union that is the second half of the Win32 `INPUT` structure. By doing it this way, I let somebody else worry about the alignment, and it'll be correct for both 32-bit and 64-bit Windows.

```
static public void Main()
{
  Console.WriteLine(Marshal.OffsetOf(typeof(INPUT), "u"));
}
```

On a 32-bit system, this prints 4, and on a 64-bit system, it prints 8. The downside is that you have to type an extra `u.` when you access the `mi`, `ki` or `hi` members.

```
input i;
i.u.mi.dx = 0;
```

(I haven't checked what the PInvoke Interop Assistant comes up with for the `INPUT` structure.)

Raymond Chen

Follow