# Why was MoveTo replaced with MoveToEx?

devblogs.microsoft.com/oldnewthing/20090720-00

July 20, 2009

Raymond Chen

Commenter Ulric asks, "Where did MoveTo(HDC, int, int) go?"

Back in the 16-bit days, the function to move the current point was called `MoveTo`, and its return value was a `DWORD` which encoded the previous position, packing two 16-bit coordinates into a single 32-bit value. As part of the transition to 32-bit Windows, GDI switched to using 32-bit coordinates instead of the wimpy 16-bit coordinates of old. As a result, it was no longer possible to encode the original position in a single `DWORD`. Something new had to be developed.

That new thing was the `MoveToEx` function. Instead of returning a single `DWORD`, it accepted a final parameter which received the previous coordinates. If you didn't care about the previous coordinates, you could just pass `NULL`. All of the GDI functions which used to pack two 16-bit coordinates into a single `DWORD` got `Ex` -ified in this way so they could accommodate the new 32-bit coordinate system.

But why did the old `MoveTo` function go away? Why not keep it around for source code compatibility?

I find this an interesting question, since most people seem to think that maintaining source code compability between the 32-bit and 64-bit versions of Windows was an idea whose stupidity rivals prosecuting a land war in Asia. (If we had followed this advice, people would just be asking, why did you replace `WinExec` with the much harder-to-use `CreateProcess`?) By the same logic, source code compatibility between 16-bit and 32-bit Windows is equally absurd. According to these people, porting 16-bit code to to 32-bit Windows is the *best time* to introduce these sorts of incompatibilities, in order to force people to rewrite their programs.

Anyway, the reason we lost `MoveTo` was that there was no way to return 64 bits of information in a 32-bit integer. Now it's true that in many cases, the caller doesn't actually care about the previous position, but of course the `MoveTo` function doesn't know that. It returns a value; it doesn't know whether the caller is going to use that return value or not.

I guess one way out would have been to change the return value of `MoveTo` to `void`. That way, people who didn't care about the return value would still compile, while people who did try to use the return value would get a compile error and have to switch to `MoveToEx`.

Yeah, I guess that could've been done, but you could also have done that yourself:

```
#define MoveTo(hdc, x, y) ((void)MoveToEx(hdc, x, y, NULL))
```

I find it interesting that most people who write their own `MoveTo` macro don't use the `(void)` cast. In most cases, this is a mistake in porting from 16-bit Windows. (I can tell because the macro is mixed in with a bunch of other porting macros.) However, in other cases, it could be intentional. The authors of the macro may simply not have known about the old 16-bit days and simply expected their macro to be used as if it were prototyped as `BOOL MoveTo(HDC, int, int)`.

These people will probably be baffled if they run across any actual 16-bit Windows code that tried to extract the high word from the return value of `MoveTo`. "Why are you extracting the high word from a `BOOL`?"

**Historical exercise**: Instead of adding a new parameter, why not just make `MoveToEx` return an `__int64`?