Why does CreateWindowEx take the extended style parameter as its first parameter instead of its last?

devblogs.microsoft.com/oldnewthing/20201207-00

December 7, 2020

Raymond Chen

Windows 3.0 expanded the `CreateWindow` function by adding a new extended style parameter, resulting in the `CreateWindowEx` function. You would expect that the new parameter would go at the end of the parameter list, but that's not where it ended up. Instead, it became a bonus *first* parameter.

```
CreateWindow   CreateWindow
(              Ex(

               dwExStyle,
lpszClass,
   lpszName,   lpszClass,
   dwStyle,       lpszName,
   x,             dwStyle,
   y,             x,
   cx,            y,
   cy,            cx,
                  cy,
hwndParent,
   hMenu,      hwndParent,
                  hMenu,
hInstance,
               hInstance,
lpCreatePara
ms             lpCreatePara
);             ms
               );
```

Why did the extra parameter go at the start rather than at the end?

If you're familiar with the `__stdcall` calling convention, you would know that the parameters are pushed onto the stack from right to left. Adding a new parameter to the front would therefore permit the old function to forward to the new function by inserting an extra parameter on the stack:

```
CreateWindow:
    pop     eax             ; pop return address
    push    0               ; dwExStyle
    push    eax             ; restore return address
    jmp     CreateWindowEx  ; continue as if CreateWindowEx
```

However, this theory doesn't hold up because Windows 3.0 used the 16-bit Pascal calling convention, which pushes parameters from left to right.

But you're close. The calling convention does play a role.

The other half of the puzzle is the in the `lParam` of the `WM_NCCREATE` and `WM_CREATE` messages. That parameter is a pointer to a `CREATESTRUCT` structure, which originally looked like this:

```
struct tagCREATESTRUCT {
  LPVOID    lpCreateParams;
  HINSTANCE hInstance;
  HMENU     hMenu;
  HWND      hwndParent;
  int       cy;
  int       cx;
  int       y;
  int       x;
  LONG      style;
  LPCSTR    lpszName;
  LPCSTR    lpszClass;
} CREATESTRUCT;
```

Look familiar?

It's the parameter list of the `CreateWindow` function, but *backward*.

Why is it backward?

Since the Pascal calling convention pushes parameters from left to right, it means that the first parameter has the highest address, and the last parameter has the lowest address. If you take all the parameters and treat them as a structure, they end up in reverse order.

And that's the missing link.

Back in the days of 16-bit Windows, the `CREATESTRUCT` that was passed to the `WM_NCCREATE` and `WM_CREATE` messages was just a pointer to the "structure" on the stack formed by all of the parameters.

For backward compatibility, the new `dwExStyle` structure member needs to go to the end, so that old code which understood the old structure would have all the old data at the old offsets.

```
struct tagCREATESTRUCT {
  LPVOID    lpCreateParams;
  HINSTANCE hInstance;
  HMENU     hMenu;
  HWND      hwndParent;
  int       cy;
  int       cx;
  int       y;
  int       x;
  LONG      style;
  LPCSTR    lpszName;
  LPCSTR    lpszClass;
  DWORD     dwExStyle; // Now with extended style support!
} CREATESTRUCT;
```

Backward compatibility dictates that the new structure member goes at the end, which means that the corresponding new parameter must go at the beginning.

It's another example of the lengths that 16-bit Windows went in order to run in a very memory-constrained system.

Bonus chatter: This means that converting a classic `CreateWindow` to the new `CreateWindowEx` is not a simply matter of inserting a new parameter under the return address. The return address plus all of the existing parameters need to be popped off, the new parameter inserted, and then all the parameters and return address pushed back on. Alternatively, the code could simply have pushed another frame onto the stack:

```
HWND CreateWindow(
   LPCSTR lpszClass,
   LPCSTR lpszName,
   DWORD dwStyle,
   int x,
   int y,
   int cx,
   int cy,
   HWND hwndParent,
   HMENU hMenu,
   HINSTANCE hInstance,
   LPVOID lpCreateParams)
{
  return CreateWindowEx(
   0,
   lpszClass,
   lpszName,
   dwStyle,
   x,
   y,
   cx,
   cy,
   hwndParent,
   hMenu,
   hInstance,
   lpCreateParams);
}
```

The code chose to do the “pop, insert, push” because the C wrapper function to push a new frame was 59 bytes long, whereas the pop/insert/push mechanism, written in hand-tuned assembly, was faster and consumed only 32 bytes.

Raymond Chen

Follow