Why is it possible to create a toolbar with the wrong HINSTANCE? And what's the right HINSTANCE anyway?

devblogs.microsoft.com/oldnewthing/20121214-00

December 14, 2012

Raymond Chen

A customer observed that all of the following code fragments are successful in creating a toolbar common control:

```
// Fragment 1: Use the process instance
    // Create the toolbar.
    HWND hWndToolbar = CreateWindowEx(
        0, TOOLBARCLASSNAME, NULL,
        WS_CHILD | TBSTYLE_WRAPABLE, 0, 0, 0, 0,
        hWndParent, NULL, g_hInst, NULL);

// Fragment 2: Use the comctl32 instance
    // Create the toolbar.
    HWND hWndToolbar = CreateWindowEx(
        0, TOOLBARCLASSNAME, L"Toolbar",
        WS_CHILD | WS_VISIBLE | WS_BORDER,
        0, 0, 0, 0,
        hWndParent, NULL, HINST_COMMCTRL, NULL);

// Fragment 3: Use NULL!
    // Create the toolbar.
    HWND hWndToolbar = CreateWindowEx(
        0, TOOLBARCLASSNAME, NULL,
        WS_CHILD | WS_VISIBLE, 0, 0, 0, 0,
        hWndParent, NULL, NULL, NULL);
```

Furthermore, the customer observed that `GetClassInfo(hinst, TOOLBARCLASSNAME, &wc)` works regardless of whether you pass the process instance or `NULL` for the `hinst` parameter.

First of all, what's going on? And second of all, which of the three methods above is most correct?

We can dispatch Fragment3 easily, because passing `NULL` as the instance handle is equivalent to passing the process instance handle. Therefore, whatever happens in Fragment-3 is explained by whatever happens in Fragment 1. (Treating a `NULL` instance as a synonym

for the process instance is a leftover behavior from 16-bit Windows, so I'm going to declare it a workaround for sloppy programming rather than a recommended practice. If you are doing this from the process module itself, then you already have your instance handle, so you should just use it. And if you are doing this from a DLL, then stop doing it, because you're messing with with somebody else's namespace.)

The behavior of Fragment 2 is easy to explain: The class is registered against the `comctl32` library, so naturally, if you create it from that library, you'll get the class.

The last case is Fragment 1: Even though we passed the wrong instance handle, we still got the control from `comctl32`. We saw the explanation for this some time ago: In order to allow the common controls classes to be used in dialog templates, they are registered as `CS_GLOBALCLASS`. One could argue that this is the recommended way of creating the window, since it allows your application to superclass a common control by registering a private class with the same name in its own namespace. Only if a custom version is not found in the provided instance is the list of global classes consulted. (I'm not saying that *I'm* arguing that position, just that it is a valid position.)

Okay, so the mystery of the instance handle has been solved. But why does `GetClassInfo` return the class even when it's registered against some other instance?

Because it found the class! `GetClassInfo` uses the same search algorithm that `CreateWindow` does, and it tells you the class it ultimately found. However, for compatibility reasons, the `WNDCLASS.hInstance` member is (usually) a copy of the `HINSTANCE` you passed to `GetClassInfo`, regardless of where the class was ultimately found.

The reason for this is that some applications pull tricks like this:

```
WNDCLASS wc;
GetClassInfo(hinstApp, "something", &wc);
... edit the WNDCLASS structure ...
UnregisterClass("something", hinstApp);
RegisterClass(&wc);
```

Suppose that `something` is a global class and suppose that the `WNDCLASS.hInstance` were set to the instance of the module that registered the global class. The application then unregisters its private class and registers what it thinks is a replacement private class. But instead, it overwrites the global class.

Oops.

The compatibility fix for this is to return `hinstApp` in the `WNDCLASS.hInstance` member. That way, these programs are tricked into registering a private class rather than overwriting a global class.

Raymond Chen

Follow