A window can have a parent or an owner but not both

devblogs.microsoft.com/oldnewthing/20100315-00

March 15, 2010

Raymond Chen

Commenter MontagFTB had a problem which, upon investigation, allegedly was caused by a subtle "fact": "The parent specified in `CreateWindowEx` is both the parent of the window and the owner of the window, but when you call `SetParent` it only sets the parent of the window, not the owner." MontagFTB then concluded that some messages were sent to the parent and others were sent to the owner.

This is a faulty diagnosis. We'll look at the correct diagnosis next time, but today's topic is parents and owners. Actually, parent and owner windows were already covered by my 2005 PDC talk, *Five Things Every Win32 Programmer Should Know*, so for most of you, today's topic is a review. (And I included the topic in the talk specifically so I wouldn't have to blog about it, but obviously that plan didn't work out.)

A window can be created as a child window (`WS_CHILD` set) or a top-level window (`WS_CHILD` not set). A child window has a parent, which you specify when you call `CreateWindowEx` , and which you can change by calling `SetParent` . A top-level window, on the other hand, has no parent. Its parent is `NULL` .

Ownership is a concept that relates top-level windows. A top-level window can optionally have an owner, which is also specified when you call `CreateWindowEx` , and which you can change by a complicated mechanism described in my talk.

Note that changing a window's parent or owner is not a normal operation. You usually create a window with a specific parent or owner and leave it that way for the lifetime of the window.

Now, a window can have a parent, or it can have an owner, or it can have neither, but it can never have both.

What would it mean for a window to have both a parent and an owner? Well, in order to have a parent, the window must itself be a child. But in order to have an owner, the window must be top-level. And top-level windows and child windows are mutually exclusive (and collectively exhaustive), because you either have the `WS_CHILD` style (which makes you a child) or you don't (which makes you top-level). Since people like tables so much, here's a table:

	Child window	Top-level window
The Parent window is…	non-`NULL`	`NULL`
The Owner window is…	N/A	`NULL` or non-`NULL`

The box for "The Owner window of a Child window…" is marked N/A because the question is meaningless. Ownership is a relationship among top-level windows.

By analogy, consider the people at a school for children. They can be separated into two groups, students and teachers. (We'll treat non-teaching staff as teachers with no students.)

Each student is assigned to a teacher. Each teacher might or might not have another teacher as a mentor. Several students can be assigned the same teacher, but every student must be assigned to some teacher. Similarly, several teachers might have the same mentor, but some teachers won't have a mentor at all, and some mentors might themselves have mentors.

It's impossible for a person to have both a teacher and a mentor, because having a teacher applies only to students, and having a mentor applies only to teachers. Teachers don't attend classes (they *lead* the classes) so they don't have a teacher. But they might have mentors. Asking for a student's mentor is a meaningless question because students don't have mentors; teachers do.

Since a window cannot have both a parent and an owner, the `CreateWindowEx` function takes a single `HWND` parameter which is either the parent or owner of the window being created, depending on what type of window you're creating. If you're creating a child window, then the parameter represents the parent window; if you're creating a top-level window, then the parameter represents the owner window.

A similar overloading of parameters happens with the `HMENU` : If you're creating a child window, then the parameter represents the child window identifier; if you're creating a top-level window, then the parameter represents the window menu. Because only top-level windows can have menus, and only child windows can have child window identifiers.

If this parameter overloading bothers you, you can write your own helper functions:

```
HWND CreateChildWindowEx(
    DWORD dwExStyle,
    LPCTSTR lpClassName,
    LPCTSTR lpWindowName,
    DWORD dwStyle,
    int x,
    int y,
    int nWidth,
    int nHeight,
    HWND hWndParent,
    UINT_PTR id,
    HINSTANCE hInstance,
    LPVOID lpParam
)
{
 // A child window must have the WS_CHILD style
 if (!(dwStyle & WS_CHILD)) {
  SetLastError(ERROR_INVALID_PARAMETER);
  return NULL;
 }
 return CreateWindowEx(
    dwExStyle,
    lpClassName,
    lpWindowName,
    dwStyle,
    x,
    y,
    nWidth,
    nHeight,
    hWndParent,
    reinterpret_cast<HMENU>(id),
    hInstance,
    lpParam);
}

HWND CreateTopLevelWindowEx(
    DWORD dwExStyle,
    LPCTSTR lpClassName,
    LPCTSTR lpWindowName,
    DWORD dwStyle,
    int x,
    int y,
    int nWidth,
    int nHeight,
    HWND hWndOwner,
    HMENU hMenu,
    HINSTANCE hInstance,
    LPVOID lpParam
)
{
 // A top-level window must not have the WS_CHILD style
 if (dwStyle & WS_CHILD) {
```

```
  SetLastError(ERROR_INVALID_PARAMETER);
  return NULL;
 }
 return CreateWindowEx(
    dwExStyle,
    lpClassName,
    lpWindowName,
    dwStyle,
    x,
    y,
    nWidth,
    nHeight,
    hWndOwner,
    hMenu,
    hInstance,
    lpParam);
}
```

There's more to parent windows and owner windows than what I've written here; I refer you to my talk (or other documentation) for more details.

Next time, we'll look at what MontagFTB is really seeing.

Raymond Chen

Follow