Why does calling SetForegroundWindow immediately followed by GetForegroundWindow not return the same window back?

devblogs.microsoft.com/oldnewthing/20161118-00

November 18, 2016

Raymond Chen

A customer said that their program used `FindWindow` to find a window in the system, then called `SetForegroundWindow` to set that window to the foreground. The call succeeds, but if they enumerate the windows to check the z-order, the window that they set as foreground is not at the top of the z-order. And if they call `GetForegroundWindow`, they don't get that window back.

So what does it mean when `SetForegroundWindow` succeeds, but doesn't actually set the foreground window?

The `SetForegroundWindow` function actually does two things, one immediately and one asynchronously.

It immediately sets the input queue associated with the window as being the foreground input queue. Among other thing, it means that keyboard input will be directed to that input queue. It also means that threads belonging to that input queue now have permission to call `SetForegroundWindow`, which is why many people affectionately call this having the foreground love.

The function also notifies the window, "Hey, you should make yourself the active window for your queue." This notification is processed synchronously if the target window's thread belongs to the same input queue as the thread that is calling `SetForegroundWindow`, and it is processed asynchronously if the window belongs to a different thread group. This notification is done by roughly the same internal nudging mechanism that threads sharing an input queue use to coordinate access to input. In particular, it means that the thread responsible for the target window needs to process messages in order to receive the nudge.

The fact that the "Go make yourself the active window for your queue" portion is asynchronous (in the cross-thread-group case) means that at the moment that `Set-ForegroundWindow` returns, the window is *becoming* the foreground window, but it is not necessarily the foreground window *yet*. If you check the z-order or call `GetForeground-Window`, you are likely to see that the target window hasn't activated yet.

Let's assume that the customer's program is doing this sort of `FindWindow` trickery as part of test automation. And let's suppose that they want the automation to wait until the target window has arrived to the foreground, so that it can continue the next step in the automation.

A bad solution would be to use the `AttachThreadInput` function to connect the test automation tool's input queue to the input queue of the target window. This is a bad solution because it means that if the target window has stopped responding, then the test automation will also stop responding. And it's bad for a test to stop responding. The purpose of the test is to monitor the main application reliably, not to get into the same jail. (Or to use a different earlier analogy, to create a joint bank account with an unreliable chap.)

What the test could do is something like this:

```
SetForegroundWindow(hwndTarget);

// Wait up to 5 seconds for the window to process the
// foreground notification.
DWORD_PTR result; // unused
if (!SendMessageTimeout(hwndTarget, WM_NULL,
                        0, 0, 0, 5000, &result)) {
    // Window was unresponsive for 5 seconds, or the
    // window was destroyed, or some other bad thing.
    ReportFailedTest();
}

if (GetForegroundWindow() != hwndTarget) {
    // The window did not become foreground for some reason.
    // Maybe there was some interference from elsewhere in the
    // system.
    ReportFailedTest();
}
```

Here we take advantage of the `WM_NULL` message. This message does nothing, so sending it has no practical effect, but the fact that we sent a message means that our code waits for the window to finish processing the *previous* message, which was "Hey, you should make yourself the active window for your queue." And that's what we are *really* waiting for.

Raymond Chen

Follow