How can I detect that my window has been suppressed from the screen by the shell?

devblogs.microsoft.com/oldnewthing/20200302-00

March 2, 2020

Raymond Chen

Some programs enumerate all windows in order to understand what is on the screen. For example, a "window picker" for a screen capture program, or a screen snipping tool, wants to find all the windows on the screen that are visible to the user so they can present them for selection.

The obvious test is `IsWindowVisible()`, which checks whether the window and all its parents have the `WS_VISIBLE` window style. If a window is not visible, then it doesn't appear on the list.

But that will still find things like windows that belong to non-current virtual desktops, or bits of the shell that are currently not visible, or some UWP apps. What's up with that?

The shell has a concept called *cloaking*, in which a window is given all the trappings of visibility, without actually being presented to the user. As far as the app can tell, it seems to be visible: It still has the `WS_VISIBLE` window style, its coordinates are still within the bounds of the monitor, it still gets `WM_PAINT` messages, it has a non-empty clipping region, all the ways a traditional Win32 app has of detecting whether it is on screen say, "Yup, you're on screen. Everything is just fine!"

If Windows didn't play this trick, it would have to make apps no longer `WS_VISIBLE` or move them off-screen or assign them an empty region or some other thing to keep them from appearing on the screen.

But the apps would be able to detect those things.

Some apps would get confused: "Well, the only time I make this window hidden is when the user is printing, and I notice that the window is now visible, so I must be printing!" Now the app thinks that it's printing when it really isn't, and it might crash because it goes to the printing component to get the name of the target printer, and the printing component says, "What are you talking about? I'm not printing!"

Some apps would notice that they had become hidden and respond by trying to get themselves visible again. “Oh no, I was moved off screen. Let me move back on screen!” Or “Oh no, I’m not visible. Let me force myself visible!”

Inventing a brand new concept (cloaking) ensures that all old apps won’t be able to detect when they were cloaked, since the concept didn’t exist at the time they were written.

But it means that if your program wants to understand all the windows that are visible to the user, you may need to incorporate cloaking into your logic.

```
BOOL IsWindowCloaked(HWND hwnd)
{
 BOOL isCloaked = FALSE;
 return (SUCCEEDED(DwmGetWindowAttribute(hwnd, DWMWA_CLOAKED,
     &isCloaked, sizeof(isCloaked))) && isCloaked;
}

BOOL IsWindowVisibleOnScreen(HWND hwnd)
{
  return IsWindowVisible(hwnd) &&
         !IsWindowCloaked(hwnd);
}
```

Raymond Chen

Follow