The case of the auto-hide taskbar

devblogs.microsoft.com/oldnewthing/20130801-00

August 1, 2013

Raymond Chen

A customer reported that their taskbar would sometimes spontaneously go into auto-hide mode. What made this particularly insidious was that they had deployed a group policy to prevent users from changing the auto-hide state (because they never wanted the taskbar to auto-hide), so when the taskbar went into auto-hide mode, there was no way to get it out of that mode!

The customer's first investigation was to find out where the auto-hide state was recorded. A little bit of registry spelunking (because as far as these people are concerned, *everything* is in the registry) showed that a single bit in the `StuckRects2` registry value controlled the auto-hide setting. They used Process Monitor and observed that it was Explorer that was updating the value. That was as far as they could troubleshoot the problem and came to the Windows team for further guidance.

It turns out that watching the registry value get updated doesn't tell you anything interesting. Explorer always writes that value when you log off, and the value written is the taskbar's current auto-hide state. The real culprit is the person who *changed* the taskbar's state, causing Explorer to save the updated state at logoff. And that culprit is somebody who called `SHAppBarMessage` with the `ABM_SETSTATE` parameter, in order to turn on the `ABS_AUTOHIDE` bit.

I warned you many years ago that the auto-hide and always-on-top states are user settings, and programs should modify them only under the instructions of the user.

The support technician was able to put together an instrumented version of the `SHAppBar-Message` function that logged any attempt to put the taskbar into auto-hide mode. (This step took a little while because the first attempt wasn't quite right and ended up not working.)

A few days later, the support technician reported back: The culprit was found with his hand in the cookie jar! One of the applications the customer was using was indeed calling `SHApp-BarMessage` with the `ABM_SETSTATE` parameter, and passing the `ABS_AUTOHIDE` flag to make the taskbar auto-hide, and the application never called the function again to restore it back to normal. Result: Taskbar goes to auto-hide and stays there.

Mind you, even if the application remember to set the auto-hide flag back to its original value, that still wouldn't have been the correct solution. Suppose two programs did this.

```
bool fWasTaskbarAutoHide;
OnStartup()
{
    fWasTaskbarAutoHide = GetTaskbarAutoHideState();
    SetTaskbarAutoHide(true);
}
OnExit()
{
    SetTaskbarAutoHide(fWasTaskbarAutoHide);
}
```

The user first runs the other program, which remembers that the taskbar is not auto-hide, then sets it to auto-hide. Now the user runs your program, which remembers that the taskbar is auto-hide, and then sets the taskbar (redundantly) to auto-hide. The user exits the first program, which sets the taskbar to normal. Now the taskbar is in normal state even though the your program wants it to be auto-hide. Finally, your program exits, and it "restores" the taskbar to auto-hide.

This is just another case of using a global setting to solve a local problem. The local solution is to create a fullscreen window, and the taskbar will get out of the way automatically.

The customer went to the online support forum for the program that was setting the taskbar to auto-hide and forgetting to restore it. And, how about that, there was a thread in that forum called something like "After I run Program X, my taskbar gets set to auto-hide."

Bonus reading: How your taskbar auto-hide settings can keep getting overwritten. It seems this problem happens a lot. This is the sort of problem you get when you decide to expose a user setting programmatically: Applications start messing with the setting when they shouldn't.

Raymond Chen

Follow