When does STARTF_USESHOWWINDOW override the parameter passed to ShowWindow()?

devblogs.microsoft.com/oldnewthing/20100301-00

March 1, 2010

Raymond Chen

kokorozashi wants to know what the rules are which govern when the second parameter to `ShowWindow` is overridden by the `STARTF_USESHOWWINDOW` flag. The guiding principle is that the parameter is ignored if the window manager thinks that the window you're creating is the application's main window. The details behind the implementation of this principle change over time, so everything from here down is implementation detail and *should not be relied upon*. I'm providing it merely to satisfy your curiosity. To reiterate, *do not rely on information in the second half of this article because it can and will change*. In fact, just to emphasize the point, I'm going to give the rules *as they once were*, not as they are today. So anybody who relies on this information is relying on implementation details of Windows which are no longer true. The window manager heuristics for determining whether the second parameter to `ShowWindow` should be overridden were once as follows: Rule zero: If the override has already been used, then don't use it again. Rule one: The easy case. If the second parameter was `SW_SHOWDEFAULT`, then the application was explicitly permitting the second parameter to `ShowWindow` to be overridden by the `STARTF_USESHOWWINDOW` flag, so let it happen. Rule two: Check the following properties.

1. The `STARTF_USESHOWWINDOW` flag was set.
2. The window was top-level.
3. The window was not owned.
4. The window had the `WS_CAPTION` style.
5. The window was not system-modal.
6. The second parameter to `ShowWindow` was `SW_SHOWNORMAL` or `SW_SHOW`.

Let's look at these heuristics one at a time. First, the `STARTF_USESHOWWINDOW` flag needed to be set: If it wasn't, then there wasn't anything to override *with*. Next, the window needed to be top-level (not a child window). Because a child window clearly is not the application's main window. The window also must not have been owned. An owned window is not the main window (the owner would be a much better candidate), and besides, it would be bad to have minimized or hidden an owned window, since that would have left the owner sitting around for apparently no reason. Even worse if the window being created was intended to be modal to the owner: You would have had a disabled window on the screen, and the window you needed to close in order to get that window enabled again was hidden! Another rule was that the window had to have a caption. This made it less likely that splash screens and other incidental windows would be misdetected as the application's main window. System-modal windows were also excluded, because you didn't want system-critical error messages to be mistaken for the application's main window. (Especially if the action was `SW_HIDE`!) The second parameter to `ShowWindow` had to be one of the special values `SW_SHOW` or `SW_SHOWNORMAL`. These values were most likely to be passed by applications which were not particular about how the window was shown. They would be comparatively unlikely to be upset that their attempt to show the window was overridden. Once a window was identified as a likely main window (either by explicitly saying so via `SW_SHOWDEFAULT` or implicitly via the heuristics), the second parameter to `ShowWindow` was ignored and replaced with the value specified by `STARTF_USESHOWWINDOW`. There was some other fiddling that happened, but they aren't really important to the topic at hand, so I'll ignore them.

Again, I reiterate that this information is provided merely to satisfy your curiosity and must not be relied upon by applications, since the heuristics may be tweaked in future versions of Windows. If you want the `STARTF_USESHOWWINDOW` flag to have an effect on your program, just pass `SW_SHOWDEFAULT` to `ShowWindow`. That's the value that says, "No really, I'm asking for it. Lemme have it."

Raymond Chen

Follow