# When programs assume that the system will never change, episode 3

devblogs.microsoft.com/oldnewthing/20060109-27

January 9, 2006

Raymond Chen

One of the stranger application compatibility puzzles was solved by a colleague of mine who was trying to figure out why a particular program couldn't open the Printers Control Panel. Upon closer investigation, the reason became clear. The program launched the Control Panel, used `FindWindow` to locate the window, then accessed that window's "File" menu and extracted the strings from that menu looking for an item that contained the word "Printer". It then posted a `WM_COMMAND` message to the Control Panel window with the menu identifier it found, thereby simulating the user clicking on the "Printers" menu option.

With Windows 95's Control Panel, this method fell apart pretty badly. There is no "Printers" option on the Control Panel's File menu. It never occurred to the authors of the program that this was a possibility. (Mind you, it was a possibility even in Windows 3.1: If you were running a non-English version of Windows, the name of the Printers option will be something like "Skrivare" or "Drucker". Not that it mattered, because the "File" menu will be called something like "Arkiv" or "Datei"! The developers of this program simply assumed that everyone in the world speaks English.)

The code never checked for errors; it plowed ahead on the assumption that everything was going according to plan. The code eventually completed its rounds and sent a garbage `WM_COMMAND` message to the Control Panel window, which was of course ignored since it didn't match any of the valid commands on that window's menu.

The punch line is that the mechanism for opening the Printers Control Panel was rather clearly spelled out on the very first page of the "Control Panel" chapter of the Windows 3.1 SDK:

> The following example shows how an application can start Control Panel and the Printers application from the command line by using the WinExec function:
>
> ```
> WinExec("control.exe printers", SW_SHOWNORMAL);
> ```

In other words, they didn't even read past the first page.

The solution: Create a "decoy" Control Panel window with the same class name as Windows 3.1, so that this program would find it. The purpose of these "decoys" is to draw the attention of the offending program, taking the brunt of the mistreatment and doing what they can to mimic the original behavior enough to keep that program happy. In this case, it waited patiently for the garbage `WM_COMMAND` message to arrive and dutifully launched the Printers Control Panel.

Nowadays, this sort of problem would probably have been solved with the use of a shim. But this was back in Windows 95, where application compatibility technology was still comparatively immature. All that was available at the time were application compatibility flags and hot-patching of binaries, wherein the values are modified as they are loaded into memory. Using hot-patching technology was reserved for only the most extreme compatibility cases, because getting permission from the vendor to patch their program was a comparatively lengthy legal process. Patching was considered a "last resort" compatibility mechanism not only for the legal machinery necessary to permit it, but also because patching a program fixes only the versions of the program the patch was developed to address. If the vendor shipped ten versions of a program, ten different patches would have to be developed. And if the vendor shipped another version after Windows 95 was delivered to duplication, that version would be broken when Windows 95 hit the shelves.

It is important to understand the distinction between what is a documented and supported feature and what is an implementation detail. Documented and supported features are contracts between Windows and your program. Windows will uphold its end of the contract for as long as that feature exists. Implementation details, on the other hand, are ephemeral; they can change at any time, be it at the next major operating system release, at the next service pack, even with the next security hotfix. If your program relies on implementation details, you're contributing to the compatibility cruft that Windows carries around from release to release.

Over the next few days, I'll talk about other decoys that have been used in Windows.

[Somebody caught my misspelling of "Drucker" while I was fixing it! – 7:45am]

Raymond Chen

**Follow**