Why are the rules for GetWindowText so weird?

devblogs.microsoft.com/oldnewthing/20030904-00

September 4, 2003

Raymond Chen

Joel Spolsky rightly points out that the rules for `GetWindowText` exhibit abstraction leakage. Why are the rules for `GetWindowText` so weird?

Set the wayback machine to 1983. Your typical PC had an 8086 processor running at a whopping 4.7MHz, two 360K 5¼-inch floppy drives (or if you were really loaded, one floppy drive and a 10MB hard drive), and 256KB of memory. [*Original entry said 256MB – oops. Thanks to Joe Beda for pointing this out.*]

This was the world of Windows 1.0.

Windows 1.0 was a coöperatively-multitasked system. No pre-emptive multitasking here. When your program got control, it had control for as long as it wanted it. Only when you called a function like `PeekMessage` or `GetMessage` did you release control to other applications.

This was important, because in the absence of a hardware memory manager, you really had to make sure that your memory didn't get ripped out from under you.

One important consequence of coöperative multitasking is that if your program is running, not only do you know that no other program is running, but you also know that **every window is responding to messages**. Why? Because if they were hung, they wouldn't have released control to you!

This means that it is **always** safe to send a message. You never had to worry about the possibility of sending a message to a hung window, since you knew that no windows were hung.

In this simpler world, `GetWindowText` was a straightforward function:

```
int WINAPI
GetWindowText(HWND hwnd, LPSTR pchBuf, int cch)
{
    // ah for the simpler days
    return SendMessage(hwnd, WM_GETTEXT, (WPARAM)cch, (LPARAM)pchBuf);
}
```

This worked for all windows, all the time. No special handling of windows in a different process.

It was the transition to Win32 and pre-emptive multitasking that forced the change in the rules, because for the first time, there was the possibility that (gasp) the window you were trying to communicate with was not responding to messages.

Now you have the backwards compatibility problem. As I described in my original article, many parts of the system and many programs rely on the ability to retrieve window text without hanging. So how do you make it possible to retrieve window text without hanging, while still giving controls like the edit control the ability to do their own window text management?

The Win32 rules on `GetWindowText` are the result of this attempt to reconcile conflicting goals.

(This same story, with slight changes, also works as a discussion of why DDE works the way it does. But fewer people use DDE nowadays, so the effect is not as dramatic.)

Raymond Chen