The secret life of GetWindowText

devblogs.microsoft.com/oldnewthing/20030821-00

August 21, 2003

Raymond Chen

GetWindowText() is more complicated than you think. The documentation tries to explain its complexity with small words, which is great if you don't understand long words, but it also means that you're not getting the full story.

Here's an attempt to give the full story.

How windows manage their text

There are two ways window classes can manage their text. They can either do it manually or they can let the system do it. The default is to let the system do it.

If a window class lets the system manage its text, the system will do the following:

- Default handling of the WM_NCCREATE message takes the lpWindowName parameter passed to CreateWindow/Ex and saves it in a "special place".
- Default handling of the WM_GETTEXT message retrieves the string from that "special place".
- Default handling of the WM_SETTEXT message copies the string to that "special place".

On the other hand, if a window class manages its window text manually, the system will not do any special handling, and it is the window class's responsibility to respond to the WM_GETTEXT/WM_SETTEXT messages and return/save the strings explicitly.

Frame windows typically let the system manage their window text. Custom controls typically manage their window text manually.

GetWindowText

GetWindowText has a problem: Window text needs to be readily available without hanging. FindWindow() needs to get window text in order to find a window. Task-switching applications need to get window text so they can display the window title in the switcher

window. It should not be possible for a hung application to clog up other applications. This is particularly true of the task switcher scenario.

This argues **against** sending WM_GETTEXT messages, because the target window of the WM_GETTEXT might be hung. Instead, GetWindowText should use the "special place" since that cannot be affected by hung applications.

On the other hand, GetWindowText is used to retrieve text from controls on a dialog, and those controls frequently employ custom text management. This argues **for** sending WM_GETTEXT messages, because that is the only way to retrieve custom-managed text.

So GetWindowText strikes a compromise.

- If you are trying to GetWindowText() from a window in your own process, then GetWindowText() will send the WM_GETTEXT message.
- If you are trying to GetWindowText() from a window in another process, then GetWindowText() will use the string from the "special place" and not send a message.

According to the first rule, if you are trying to get text from a window in your own process, and the window is hung, then GetWindowText() will also hang. But since the window belongs to your process, it's your own fault and you deserve to lose. Sending the WM_GETTEXT message ensures that text from windows that do custom text management (typically, custom controls) are properly retrieved.

According to the second rule, if you are trying to get text from a window in another process, then GetWindowText() will not send a message; it will just retrieve the string from the "special place". Since the most common reason for getting text from a window in another process is to get the title of the frame, and since frame windows typically do not do custom window text manipulation, this usually gets the right string.

The documentation simplifies this as "GetWindowText() cannot retrieve text from a window from another application."

What if I don't like these rules?

If the second rule bothers you because you need to get text from a custom control in another process, then you can send the WM_GETTEXT message manually. Since you are not using GetWindowText(), you are not subject to its rules.

Note, however, that if the target window is hung, your application will also hang since SendMessage() will not return until the target window responds.

Note also that since WM_GETTEXT is in the system message range (0 to WM_USER-1), you do not need to do any parameter marshalling (and in fact, you shouldn't). USER will do the marshalling for you.

Can you give an example where this makes a difference?

Consider this control:

```
SampleWndProc(...)
{
    case WM_GETTEXT:
        lstrcpyn((LPTSTR)lParam, "Booga!", (int)wParam);
        return lstrlen((LPTSTR)lParam);
    case WM_GETTEXTLENGTH: return 7; // lstrlen("Booga!") + null
    ...
}
```

Now consider application A that does

```
hwnd = CreateWindow("Sample", "Frappy", ...);
```

Now consider process B that gets the handle to the window created by application A (by whatever means).

```
TCHAR szBuf[80];
GetWindowText(hwnd, szBuf, 80);
```

This will return szBuf = "Frappy" because it is getting the window text from the "special place". However,

```
SendMessage(hwnd, WM_GETTEXT, 80, (LPARAM)szBuf);
```

will return szBuf = "Booga!"

Raymond Chen

Follow