What can I do with the HINSTANCE returned by the ShellExecute function?

devblogs.microsoft.com/oldnewthing/20060505-08

May 5, 2006

Raymond Chen

As we saw earlier, in 16-bit Windows, the `HINSTANCE` identified a program. The Win32 kernel is a complete redesign from the 16-bit kernel, introducing such concepts as “kernel objects” and “security descriptors”. In particular 16-bit Windows didn’t have “process IDs”; the instance handle served that purpose. That is why the `WinExec` and `ShellExecute` functions returned an `HINSTANCE`. But in the 32-bit world, `HINSTANCE`s do not uniquely identify a running program since it is merely the base address of the executable. Since each program runs in its own address space, that value is hardly unique across the entire system. So what can you do with the `HINSTANCE` returned by the `ShellExecute` function? You can check if it greater than 32, indicating that the call was successful. If the value is less than 32, then it is an error code. The precise value of the `HINSTANCE` in the greater-than-32 case is meaningless. Why am I bothering to tell you things that are already covered in MSDN? Because people still have trouble putting two and two together. I keep seeing people who take the `HINSTANCE` returned by the `ShellExecute` function and hunt through all the windows in the system looking for a window with a matching `GWLP_HINSTANCE` (or `GWL_HINSTANCE` if you’re still living in the unenlightened non-64-bit-compatible world). This doesn’t work for the two reasons I described above. First, the precise value of the `HINSTANCE` you get back is meaningless, and even if it were meaningful, it wouldn’t do you any good since the `HINSTANCE` is not unique. (In fact, the `HINSTANCE` for a process is nearly always 0x00400000, since that is the default address most linkers assign to program executables.) The most common reason people want to pull this sort of trick in the first place is that they want to do something with the program that was just launched, typically, wait for it to exit, indicating that the user has closed the document. Unfortunately, this plan comes with its own pitfalls. First, as we noted, the `HINSTANCE` that you get from the `ShellExecute` function is useless. You have to use the `ShellExecuteEx` function and set the `SEE_MASK_NOCLOSEPROCESS` flag in the `SHELLEXECUTEINFO` structure, at which point a handle to process is returned in the `hProcess` member. But that still doesn’t work. A document can be executed with no new process being created. The most common case (but hardly the only such) in which you will encounter this is if the registered handler for the document type requested a DDE conversation. In that case, an existing instance of the program has accepted responsibility for the document. Waiting for the process to exit is not the same as waiting for the user to close the document, because closing the document doesn't exit the process. Just because the user closes the document doesn't mean that the process exits. Most programs will let you open a new document from the "File" menu. Once that new document is opened, the user can close the old one. (Single-document programs implicitly close the old document when the new one is opened.) What's more, closing all open windows associated with the document need not result in the program exiting. Some programs run in the background even after you've closed all their windows, either to provide some sort of continuing service, or just because they are just anticipating that the user will run the program again soon so they delay the final exit for a few minutes to see if they will be needed. Just because the process exits doesn't mean that the document is closed. Some programs detect a previous instance and hand off the document to that instance. Other programs are stubs that launch another process to do the real work. In either case, the newly-created process exits quickly, but the document is still open, since the responsibility for the document has been handed off to another process.

There is no uniform way to detect that a document has been closed. Each program handles it differently. If you're lucky, the program exposes properties that allow you to monitor the status of an open document. As we saw earlier, Internet Explorer exposes properties of its open windows through the `ShellWindows` object. I understand that Microsoft Office also exposes a rather elaborate set of automation interfaces for its component programs.

Raymond Chen

Follow