How do I test that return value of ShellExecute against 32?

devblogs.microsoft.com/oldnewthing/20061108-05

November 8, 2006

Raymond Chen

We discussed earlier the history behind the the return value of the `ShellExecute` function, and why its value in Win32 is meaningless aside from testing it against the value 32 to determine whether an error occurred.

How, then, should you check for errors?

Let's turn the question around. How would you, the implementor of the `ShellExecute` function, report success? The `ShellExecute` is a very popular function, so you have to prepared for the ways people check the return code incorrectly yet manage to work in spite of themselves. The goal, therefore, is to report success in a manner that breaks as few programs as possible.

(Now, there may be those of you who say, "Hang compatibility. If programs checked the return value incorrectly, then they deserve to stop working!" If you choose to go in that direction, then be prepared for the deluge of compatibility bugs to be assigned to you to fix. And they're going to come from a grumpy compatibility testing team because they will have spent a long time just finding out that the problem was that the program was checking the return value of `ShellExecute` incorrectly.)

Since there is still 16-bit code out there that may thunk up to 32-bit code, you probably don't want to return a value greater than `0xFFFF`. Otherwise, when that value gets truncated to a 16-bit `HINSTANCE` will lose the high word. If you returned a value like `0x00010001`, this would truncate to `0x0001`, which would be treated as an error code.

For similar reasons, the 64-bit implementation of the `ShellExecute` function had better not use the upper 32 bits of the return value. Code that casts the return value to `int` will lose the high 32 bits.

Furthermore, you shouldn't return a value that, when cast to an integer, results in a negative number. Some people will use a signed comparison against 32; others will use an unsigned comparison. If you returned a value like `-5`, then the people who used a signed comparison

would think the function failed, whereas those who used an unsigned comparison would think it succeeded.

By the same logic, the value you choose as the return value should not result in a negative number when cast to a 16-bit integer. If the return value is passed to a 16-bit caller that casts the result to an integer and compares against 32, you want consistent results independent of whether the 16-bit caller used a signed or unsigned comparison.

Edge conditions are tricky, so you don't want to return the value 32 exactly. If you look at code that checks the return value from `ShellExecute`, you'll probably find that the world is split as to whether 32 is an error code or not. So it'd be in your best interest not to return the value 32 exactly but rather a value larger than 32.

So far, you're constrained to choosing a value in the range 33–32767.

Finally, you might be a fan of Douglas Adams. (Most geeks are.) The all-important number 42 fits into this range. Your choice of return value, therefore, might be `(HINSTANCE)42`.

Going back to the original question: How should I check the return value of `ShellExecute` for errors? MSDN says you can cast the result to an integer and compare the result against 32. That'll work fine. You could cast in the other direction, comparing the return value against `(HINSTANCE)32`. That'll work fine, too. Or you could cast the result to an `INT_PTR` and compare the result against 32. That's fine, too. They'll all work, because the implementor of the `ShellExecute` function had to plan ahead for you and all the other people who call the `ShellExecute` function.

Raymond Chen

Follow