What are the conventions for managing standard handles?

devblogs.microsoft.com/oldnewthing/20130307-00

March 7, 2013

Raymond Chen

Consider this function:

```
void ChangeConsoleColor(WORD wColor)
{
 HANDLE h = GetStdHandle(STD_OUTPUT_HANDLE);
 if (h != INVALID_HANDLE_VALUE) {
  SetConsoleTextAttribute(h, wColor);
  CloseHandle(h);
 }
}
```

"When I call this function, it works the first time, but when I call it a second time, `GetStdHandle` returns a handle numerically identical to the one returned by the first call, but the handle is now invalid, presumably because I closed it. I closed it because I was taught to clean up after myself. Is this a case where I shouldn't?"

Yes, you should clean up after yourself, but you should also have been taught to be respectful of community property. In this case, you walked into the TV room of your dormitory, watched an episode of *Friends*, and then smashed the television with a baseball bat. Later, you came back to the room to watch another episode of *Friends* and said, "Hey, what happened to our television?" (You can tell I'm old because I'm talking about the TV room of a dormitory.)

The standard handle values are sort of like a global variable for your process. Anybody can call `GetStdHandle` to read the variable, and anybody can call `SetStdHandle` to set it. But as with any other global handle variable, you need to observe certain rules to ensure that the value is always valid.

Suppose you had a global variable called `HANDLE hSomeFile`. What invariants would you want to apply?

- If the value is `INVALID_HANDLE_VALUE`, then there is no active file. (You might also have decided to use `NULL` as your special value, but `INVALID_HANDLE_VALUE` works better here because that is the conventional sentinel value for file handles.)
- If the value is not the special value above, then it refers to a valid file handle.

That second invariant above already establishes a rule:

> If you close the handle held in the global variable, you must also set the global variable to a new valid value.

As I noted some time ago, programming is a game of stepping-stone from one island of consistency to another. You start with a consistent system, you perturb it (temporarily violating consistency), and then you re-establish consistency. Closing the handle makes the value invalid, so you need to follow up by making the value valid again. Otherwise you left your system in an inconsistent state.

Okay, now instead of talking about that global variable `hSomeFile`, let's talk about the global handle hidden behind `GetStdHandle` and `SetStdHandle`. Congratulations, we just established the rules for managing standard handles.

- If `GetStdHandle` returns `INVALID_HANDLE_VALUE`, then there is no active file.
- If the value is not the special value above, then it refers to a valid file handle. (Note that file handles can refer to things that aren't files. In our case, it often will refer to a console.)
- If you call `CloseHandle` on a standard handle, then you must also call `SetStdHandle` to set a new value for the standard handle.

Note that these rules are just conventions. If you want to violate them by, say, closing the handle and then leaving a garbage handle in the hidden global variable for the next guy to trip over, then that's your problem. For example, you might choose to violate the rules temporarily, and then fix things up before anybody notices.

Raymond Chen

Follow