Why are HANDLE return values so inconsistent?

devblogs.microsoft.com/oldnewthing/20040302-00

March 2, 2004

Raymond Chen

If you look at the various functions that return `HANDLE`s, you'll see that some of them return `NULL` (like `CreateThread`) and some of them return `INVALID_HANDLE_VALUE` (like `CreateFile`). You have to check the documentation to see what each particular function returns on failure.

Why are the return values so inconsistent?

The reasons, as you may suspect, are historical.

The values were chosen to be compatible with 16-bit Windows. The 16-bit functions `OpenFile`, `_lopen` and `_lcreat` return `-1` on failure, so the 32-bit `CreateFile` function returns `INVALID_HANDLE_VALUE` in order to facilitate porting code from Win16.

(Armed with this, you can now answer the following trivia question: Why do I call `CreateFile` when I'm not actually creating a file? Shouldn't it be called `OpenFile`? Answer: Yes, `OpenFile` would have been a better name, but that name was already taken.)

On the other hand, there are no Win16 equivalents for `CreateThread` or `CreateMutex`, so they return `NULL`.

Since the precedent had now been set for inconsistent return values, whenever a new function got added, it was a bit of a toss-up whether the new function returned `NULL` or `INVALID_HANDLE_VALUE`.

This inconsistency has multiple consequences.

First, of course, you have to be careful to check the return values properly.

Second, it means that if you write a generic handle-wrapping class, you have to be mindful of two possible "not a handle" values.

Third, if you want to pre-initialize a `HANDLE` variable, you have to initialize it in a manner compatible with the function you intend to use. For example, the following code is wrong:

```
HANDLE h = NULL;
if (UseLogFile()) {
    h = CreateFile(...);
}
DoOtherStuff();
if (h) {
   Log(h);
}
DoOtherStuff();
if (h) {
    CloseHandle(h);
}
```

This code has two bugs. First, the return value from `CreateFile` is checked incorrectly. The code above checks for `NULL` instead of `INVALID_HANDLE_VALUE`. Second, the code initializes the `h` variable incorrectly. Here's the corrected version:

```
HANDLE h = INVALID_HANDLE_VALUE;
if (UseLogFile()) {
    h = CreateFile(...);
}
DoOtherStuff();
if (h != INVALID_HANDLE_VALUE) {
   Log(h);
}
DoOtherStuff();
if (h != INVALID_HANDLE_VALUE) {
    CloseHandle(h);
}
```

Fourth, you have to be particularly careful with the `INVALID_HANDLE_VALUE` value: By coincidence, the value `INVALID_HANDLE_VALUE` happens to be numerically equal to the pseudohandle returned by `GetCurrentProcess()`. Many kernel functions accept pseudohandles, so if if you mess up and accidentally call, say, `WaitForSingleObject` on a failed `INVALID_HANDLE_VALUE` handle, you will actually end up waiting on your own process. This wait will, of course, never complete, because a process is signalled when it exits, so you ended up waiting for yourself.

Raymond Chen

Follow