# When will GetMessage return -1?

devblogs.microsoft.com/oldnewthing/20130322-00

March 22, 2013

Raymond Chen

A source of great consternation is the mysterious `-1` return value from `GetMessage`:

> If there is an error, the return value is −1. For example, the function fails if `hWnd` is an invalid window handle or `lpMsg` is an invalid pointer.

That paragraph has caused all sorts of havoc, because it throws into disarray the standard message pump:

```
MSG msg;
while (GetMessage(&msg, NULL, 0, 0)) {
 ...
}
```

But don't worry, the standard message pump is safe. If your parameters are exactly

- a valid pointer to a valid `MSG` structure,
- a null window handle,
- no starting message range filter,
- no ending message range filter,

then `GetMessage` will not fail with `-1`.

Originally, the `GetMessage` function did not have a failure mode. If you passed invalid parameters, then you invoked undefined behavior, and you probably crashed.

Later, somebody said, "Oh, no, the `GetMessage` function needs to detect invalid parameters and instead of crashing, it needs to fail gracefully with some sort of error code." (This was before "Fail-Fast" came into fashion.)

The problem is that `GetMessage`'s return value of `BOOL` was already specified not as a success/failure code, but rather a "Has a `WM_QUIT` message been received?" code. So return `FALSE` wouldn't work.

The solution (if that's what you want to call it) was to have `GetMessage` return the not-really-a-`BOOL`-but-we'll-pretend-it-is value `-1` to signal an *invalid parameter* error.

And that's what threw everybody into a tizzy, because now every message loop looks buggy.

But you can calm down. The standard message loop is fine. All the parameters are hard-coded (and therefore valid by inspection), save for the `&msg` parameter, which is still valid by inspection. So that case is okay. It has to be, for compatibility.

The people who need to worry are people who pass a variable as the window handle filter (because that window handle may no longer be valid), or pass dynamically-allocated memory as the `lpMsg` (because the pointer may no longer be valid), or who pass a nontrivial message filter (because the filter parameters may be invalid).

In practice, the memory for the `lpMsg` is nearly always a stack variable (so the pointer is valid), and the message range filters are hard-coded (so valid by inspection). The one to watch out for is the window handle filter. But we saw earlier that a filtered `GetMessage` is a bad idea anyway, because your program will not respond to messages that don't meet the filter.

Raymond Chen

**Follow**