What does it mean when it says that FORMAT_MESSAGE_ALLOCATE_BUFFER can be used in Store apps starting in Windows 10?

devblogs.microsoft.com/oldnewthing/20200805-00

August 5, 2020

Raymond Chen

Tucked away in the documentation for the `FormatMessage` function is this little note:

> In previous versions of Windows, this value was not available for use when compiling Windows Store apps. As of Windows 10 this value can be used.

What does this even mean?

And what happens if I try to use this flag in a Windows Store app that sets its minimum operating system as Windows 8?

Okay, first of all, let's set the context. The context is Windows Store apps; that is, apps that are submitted to the Windows Store and which run as a UWP app. These apps must pass a test known as the Windows App Certification Kit or just WACK for short. Passing this test as a prerequisite for being accepted by the Windows Store is a *policy* issue, not a technical one.

In Windows 8, programs which used the `FORMAT_ MESSAGE_ ALLOCATE_ BUFFER` flag were not accepted by WACK. To decrease the possibility of a program accidentally using the flag, only to discover at submission time that it was using a forbidden flag, the Windows header files checked whether they were being include by a program that intended to be a UWP app. If so, then the definition of the `FORMAT_ MESSAGE_ ALLOCATE_ BUFFER` flag was `#ifdef` 'd out. That way, if you tried to use the flag, you got an *undefined identifier* error at compile time.[1]

What the text is trying to say is that the Windows Store changed their policy so that programs which use the `FORMAT_ MESSAGE_ ALLOCATE_ BUFFER` flag will be accepted. This change in policy happened to occur at exactly the same time that Windows 10 was released.

This was not a coincidence.

The Windows 10 SDK came with an updated version of WACK that permitted the `FORMAT_ MESSAGE_ ALLOCATE_ BUFFER` flag to be used. Furthermore, the Windows 10 SDK header files removed the `#ifdef` around the definition of the `FORMAT_ MESSAGE_ ALLOCATE_ BUFFER` flag, so that the flag became visible.

Can you use that flag in an app that is marked as backward compatible to Windows 8?

Sure.

Will it pass WACK?

It does now.[2]

What was so horrible about the `FORMAT_ MESSAGE_ ALLOCATE_ BUFFER` flag that the Windows Store disallowed it for so long? We'll look at that next time.

[1] Of course, there's nothing stopping you from defining the value yourself and using it. That will get your code to compile, but you would still have to deal with WACK. The reason for removing the definition is to help you find your mistakes early, not to stop you from making mistakes altogether.

[2] The Store does not run an old version of WACK if your program was designed for an older version of Windows. It always runs the latest version of WACK regardless of your target. The Windows SDK contains the version of WACK which was current at the time the SDK was published, and that may have contributed to the confusion, making people think that the Store ran the version of WACK that matched the SDK the program used. It doesn't. The Store always runs the latest WACK.

Raymond Chen

Follow