Why does GetServiceDisplayNameA report a larger required buffer size than actually necessary?

devblogs.microsoft.com/oldnewthing/20180608-00

June 8, 2018

Raymond Chen

If you call the `GetServiceDisplayNameA` function (the ANSI version), it reports a buffer size far larger than what appears to be necessary.

For example, if the service display name is `awesome`, you would expect that the required buffer size is seven characters, one for each character in `"awesome"`. (The `GetService-DisplayNameA` function does not count the terminating null; it expects you to add one yourself.)

But instead, the `GetServiceDisplayNameA` function says that you need fourteen characters. And then when you give it a buffer that is fifteen characters long, it fills in only eight of them, and then says "Ha ha, I wrote only seven characters (plus the terminating null). Silly you allocated far more memory than you needed to, sucka!"

Why is it reporting a required buffer size larger than what it actually needs?

Because character set conversion is hard.

When you call the `GetServiceDisplayNameA` function (ANSI version), it forwards the call to `GetServiceDisplayNameW` function (Unicode version). If the Unicode version says, "Sorry, that buffer is too small; it needs to be big enough to hold *N* Unicode characters," the ANSI version doesn't know how many ANSI characters that translates to. A single Unicode character could expand to as many as two ANSI characters in the case where the ANSI code page is DBCS. The `GetServiceDisplayNameA` function plays it safe and takes the worst-case scenario that the service display name consists completely of Unicode characters which require two ANSI characters to represent.

That's why it over-reports the buffer size.

When you call it with a buffer that is fifteen characters long, the `GetServiceDisplayNameA` function calls the `GetServiceDisplayNameW` function, which says, "No problem, here's your display name. It's seven Unicode characters long." The `GetServiceDisplayNameA` function then converts it from Unicode to ANSI, and it turns out that it requires only seven characters plus the terminating null. Hm, how about that. Okay, well here’s your seven-character string. Sorry about the extra seven characters you allocated. I asked you to allocate them just in case.

Bonus chatter: These worst-case calculations will break if the ANSI code page were ever UTF-8, because the worst-case expansion becomes three UTF-8 code units for one UTF-16 code unit, rather than just two to one for DBCS code pages. These types of assumptions about the worst-case scenario are buried throughout tens of millions of lines of source code. Finding them is quite a challenge.

Raymond Chen

Follow