What is the correct way of using the string buffer returned by the WindowsPreallocateStringBuffer function?

devblogs.microsoft.com/oldnewthing/20170913-00

September 13, 2017

Raymond Chen

The most common way of creating an `HSTRING` is to call `WindowsCreateString`, but there is also a two-phase creation pattern: First you call `WindowsPreallocateStringBuffer` to create a buffer for a future string. You then fill the buffer with stringy goodness and then call `WindowsPromoteStringBuffer` to convert it to a real `HSTRING`. (Or you can call `WindowsDeleteStringBuffer` to change your mind and pretend it never happened.)

The rule for managing the buffer returned by `WindowsPreallocateStringBuffer` is that you are expected to write *exactly* `length` code units into the buffer. No more. No less. The system already put a terminating null after the end of the buffer; your job is to fill the buffer with the string contents.

For example, if you want to use two-phase creation to create the string `hello`, you would call `WindowsPreallocateStringBuffer` and pass `length = 5`. Into the resulting buffer, you write the characters `h`, `e`, `l`, `l`, and `o`, and that's all. The system already stored the terminating null.

This particular formulation of the rules is important in the case that `length = 0`.[1] Since the representation of an `HSTRING` of length zero is the null pointer, there is no actual buffer. What happens is that the system uses a single preallocated buffer (consisting of just a null terminator) to represent the buffer for all zero-length strings. If you call `WindowsPreallocateStringBuffer`, you get a pointer to that preallocated buffer.[2] Since you passed a length of zero, you are expected to write zero characters to the buffer; in other words, you are expected to do nothing at all with the buffer.

And of course since `HSTRING`s are immutable, your permission to modify the buffer ends once you promote the buffer to a string. Once it's been promoted to a string, the entire buffer becomes read-only.

[1] Another way of interpreting this corner case is to say "Don't bother calling `WindowsPreallocateStringBuffer` with a string of length zero. Otherwise, go ahead and call it, and you can write that null terminator if you like."

[2] Arguably, to accommodate the possibiltiy of somebody writing that null terminator, it should return a preallocated *writable* buffer just large enough to hold that null terminator. It could be the high 16 bits of the `length` field itself!

Raymond Chen

Follow