Why can't I PostMessage the WM_COPYDATA message, but I can SendMessageTimeout it with a tiny timeout?

devblogs.microsoft.com/oldnewthing/20110916-00

September 16, 2011

Raymond Chen

After receiving the explanation of what happens to a sent message when `SendMessageTimeout` reaches its timeout, a customer found that the explanation raised another question: If the window manager waits until the receiving thread finishes processing the message, then why can't you post a `WM_COPYDATA` message? "After all, `SendMessageTimeout` with a very short timeout isn't all that different from `PostMessage`." Actually, `SendMessageTimeout` with a very short timeout is completely different from `PostMessage`. Let's set aside the one crucial difference that, unlike messages posted by `PostMessage`, which cannot be recalled, the `SendMessageTimeout` function will cancel the message entirely if the receiving thread does not process messages quickly enough. Recall that messages posted to a queue via `PostMessage` are retrieved by the `GetMessage` function and placed in a `MSG` structure. Once that's done, the window manager disavows any knowledge of the message. It did its job: It placed the message in the message queue and produced it when the thread requested the next message in the queue. What the program does with the message is completely up in the air. There's no metaphysical requirement that the message be dispatched to its intended recipient. (In fact, you already know of a common case where messages are "stolen" from their intended recipients: Dialog boxes.) In principle, the message pump could do anything it wants to the message. Dispatch it immediately, steal the message, throw the message away, eat the message and post a different message, even save the message in its pocket for a rainy day. By contrast, there's nothing you can do to redirect inbound non-queued messages. They always go directly to the window procedure. The important difference from the standpoint of messages like `WM_COPYDATA` is that with sent messages, the window manager knows when message processing is complete: When the window procedure returns. At that time, it can free the temporary buffers used to marshal the message from the sender to the recipient. If the message were posted, the window manager would never be sure. Suppose the message is placed in a `MSG` structure as the result of a call to `GetMessage`. Now the window manager knows that the receiving thread has the potential for taking action on the message and the buffers need to be valid. But how would it know when the buffers can be freed? "Well you can wait until the exact same parameters get passed in a `MSG` structure to the `DispatchMessage` function." But what if the message loop discards the message? Or what if it decides to dispatch it twice? Or what if it decides to smuggle it inside another message?

Posted messages have no guarantee of delivery nor do they provide any information as to when the message has been definitely processed, or even if it has been processed at all. If the window manager let you post a `WM_COPYDATA` message, it would have to use its psychic powers to know when the memory can be freed.

Raymond Chen

Follow