# Using WM_COPYDATA to marshal message parameters since the window manager otherwise doesn't know how

devblogs.microsoft.com/oldnewthing/20121019-00

October 19, 2012

Raymond Chen

Miral asks for the recommended way of passing messages across processes if they require custom marshaling.

There is no one recommended way of doing the custom marshaling, although some are hackier than others.

Probably the most *architecturally beautiful* way of doing it is to use a mechanism that *does* perform automatic marshaling, like COM and MIDL. Okay, it's not actually automatic, but it does allow you just give MIDL your structures and some information about how they should be interpreted, and the MIDL compiler autogenerates the marshaler. You can then pass the data back and forth by simply invoking COM methods and letting COM do the work.

*Architecturally beautiful* often turns into *forcing me to learn more than I really wanted to learn*, so here's a more self-contained approach: Take advantage of the `WM_COPYDATA` message. This is sort of the poor-man's marshaler. All it knows how to marshal is a blob of bytes. It's your responsibility to take what you want to marshal and serialize it into a blob of bytes. `WM_COPYDATA` will get the bytes to the other side, and then the recipient needs to deserialize the blob of bytes back into your data. But at least `WM_COPYDATA` does the tricky bit of getting the bytes from one side to the other.

Let's start with our scratch program and have it transfer data to another copy of itself. Make the following changes:

```
#include <strsafe.h>

HWND g_hwndOther;

#define CDSCODE_WINDOWPOS 42 // lpData -> WINDOWPOS

void OnWindowPosChanged(HWND hwnd, LPWINDOWPOS pwp)
{
 if (g_hwndOther) {
  COPYDATASTRUCT cds;
  cds.dwData = CDSCODE_WINDOWPOS;
  cds.cbData = sizeof(WINDOWPOS);
  cds.lpData = pwp;
  SendMessage(g_hwndOther, WM_COPYDATA,
            reinterpret_cast<WPARAM>(hwnd),
            reinterpret_cast<LPARAM>(&cds));
 }
 FORWARD_WM_WINDOWPOSCHANGED(hwnd, pwp, DefWindowProc);
}

void OnCopyData(HWND hwnd, HWND hwndFrom, PCOPYDATASTRUCT pcds)
{
 switch (pcds->dwData) {
 case CDSCODE_WINDOWPOS:
  if (pcds->cbData == sizeof(WINDOWPOS)) {
   LPWINDOWPOS pwp = static_cast<LPWINDOWPOS>(pcds->lpData);
   TCHAR szMessage[256];
   StringCchPrintf(szMessage, 256,
    TEXT("From window %p: x=%d, y=%d, cx=%d, cy=%d, flags=%s %s"),
    hwndFrom, pwp->x, pwp->y, pwp->cx, pwp->cy,
    (pwp->flags & SWP_NOMOVE) ? TEXT("nomove") : TEXT("move"),
    (pwp->flags & SWP_NOSIZE) ? TEXT("nosize") : TEXT("size"));
   SetWindowText(hwnd, szMessage);
  }
  break;
 }
}

// WndProc

    HANDLE_MSG(hwnd, WM_WINDOWPOSCHANGED, OnWindowPosChanged);
    HANDLE_MSG(hwnd, WM_COPYDATA, OnCopyData);

// WinMain
    // If there is another window called "Scratch", then it becomes
    // our recipient.
    g_hwndOther = FindWindow(TEXT("Scratch"), TEXT("Scratch"));

    hwnd = CreateWindow(
        "Scratch",                      /* Class Name */
        g_hwndOther ? TEXT("Sender") : TEXT("Scratch"),
        WS_OVERLAPPEDWINDOW,            /* Style */
```

```
    CW_USEDEFAULT, CW_USEDEFAULT,   /* Position */
    CW_USEDEFAULT, CW_USEDEFAULT,   /* Size */
    NULL,                           /* Parent */
    NULL,                           /* No menu */
    hinst,                          /* Instance */
    0);                             /* No special parameters */
```

Just to make it easier to tell the two windows apart, I call the one sending the message "Sender". (Note that my method for finding the other window is pretty rudimentary, because that's not the point of the example.)

Whenever the sender window receives a `WM_WINDOWPOSCHANGED` message, it sends a copy of the `WINDOWPOS` structure to the recipient, which then displays it in its own title bar. Things to observe:

- The value you put into `dwData` can be anything you like. It's just another `DWORD` of data. Traditionally, it's used like a "message number", used to communicate what type of data is being sent. In our case, we choose 42 to mean "The `lpData` points to a `WINDOWPOS` structure."
- The `cbData` is the number of bytes you want to send, and `lpData` points to the buffer. In our case, the number of bytes is always the same, but variable-sized data is also fine.
- The `lpData` can point anywhere, as long as the memory is valid for the lifetime of the `SendMessage` call. In this case, I just point it at the data given to me by the window manager. Of course, if you allocated memory to put into `lpData`, then the responsibility for freeing it also belongs to you.
- For safety's sake, I validate that when I get a `CDSCODE_WINDOWPOS` request, the associated data really is the size of a `WINDOWPOS` structure. This helps protect against a rogue caller who tries to crash the application by sending a `CDSCODE_WINDOWPOS` with a size less than `sizeof(WINDOWPOS)`, thereby triggering a buffer overflow. (Exercise: Under what other conditions can the size be incorrect? How would you fix that?)
- The `WM_COPYDATA` copies data in only one direction. It does not provide a way to pass information back to the sender. (Exercise: How would you pass information back?)
- The `hwndFrom` parameter is a courtesy parameter, like `dwData`. There is currently no attempt to verify that the window really is that of the sender. (In practice, all that could really be verified is that the window belongs to the thread that is doing the sending, but right now, not even that level of validation is performed.)

The `WM_COPYDATA` message is suitable for small-to-medium-sized amounts of memory. Though if the amount of memory is so small that it fits into a `WPARAM` and `LPARAM`, then even `WM_COPYDATA` is overkill.

If you're going to be passing large chunks of memory, then you may want to consider using a shared memory handle instead. The shared memory handle also has the benefit of being shared, which means that the recipient can modify the shared memory block, and the sender can see the changes. (Yes, this is one answer to the second exercise, but see if you can find another answer that tays within the spirit of the exercise.)

Raymond Chen

**Follow**