What is the hSection parameter to CreateDIBSection for?

devblogs.microsoft.com/oldnewthing/20100108-00

January 8, 2010

Raymond Chen

The `CreateDibSection` function creates a special type of bitmap known as a DIB section. We've worked with these guys before: The feature of DIB sections that is by far the most interesting is that the raw pixels in the bitmap are mapped into your process space as if they were normal memory, which you can read from and write to directly.

> But what is the deal with that funky `hSection` parameter? Although the `ppvBits` parameter receives "a pointer to the location of the DIB bit values," the documentation also says that if you pass `NULL`, then "an application cannot later obtain a handle to this memory." That second part makes no sense. Why would I want to obtain a handle to the memory if I passed `NULL`, since I told it I didn't have any memory to begin with? Why is it so important to call out that I can't retrieve `NULL`? The documentation appears to go out of its way to point out something that makes no sense!

Let's look at that second part in a little more context. Here is the entire description for the `hSection` parameter:

> If *hSection* is NULL, the system allocates memory for the DIB. In this case, the **CreateDIBSection** function ignores the *dwOffset* parameter. An application cannot later obtain a handle to this memory. The **dshSection** member of the **DIBSECTION** structure filled in by calling the **GetObject** function will be NULL.

The "this" in "this memory" is "the memory the system allocated," not "the memory the application passed in." Because, as you noticed, the application didn't pass in any memory! It's trying to tell you that when you tell the system to allocate memory for the DIB section, you don't get to peek back in and get the handle to the memory which the system allocated. Let's look at the bigger picture here. The memory for a DIB section needs to be mapped into the application's address space, and from the description, the application has the option of passing explicit storage in the form of a file mapping handle (and offset into that file mapping handle), or it can pass `NULL` and let GDI worry about where to store it. If you were writing the GDI code to manage the memory for DIB sections, how would you do it? *How would you do it?* is is another one of those problem-solving questions similar to *What would the world be like if that were true?* or *Imagine if two programs did this?* For one thing, *How would*

you do it? lets you rule out designs that involve clairvoyance or psychic powers. (It's surprising how often people quietly assume that systems are built upon clairvoyance.) For another thing, it forces you to put on a different hat and view the problem from another point of view, one which may help you understand the system better. If you had to implement a system where memory could be managed either in the form of a file mapping handle or a mechanism of your choice, you probably would choose as your alternate mechanism… a file mapping handle. That way, there is only one code path for memory management instead of two. Suppose you had to install a door security system that both you and Bob could use. Bob insists that he use a traditional metal key to unlock the door, but says you can use any system you want. Would you design a combination system that could be unlocked either with a metal key or an electronic smart card? Or would you just use a traditional keyed lock with two sets of keys, giving one of Bob and keeping the other for yourself? When you create a DIB section and pass `NULL` for the `hSection`, GDI simply creates an internal `hSection` and uses that. The documentation is trying to say that the internal `hSection` is inaccessible to the application.

Note: I don't know if this is actually what happens internally, but it's the simplest explanation that matches the known facts.

Raymond Chen

Follow