A survey of the various ways of creating GDI bitmaps with predefined data

devblogs.microsoft.com/oldnewthing/20170331-00

March 31, 2017

Raymond Chen

Suppose you have a buffer full of pixels and you want to create a GDI bitmap that contains those pixels.

We'll start with the `CreateBitmap` function. This creates a device-dependent bitmap with the pixels you provide. The weird thing about this function is that it's a device-dependent bitmap, but there's no parameter that specifies the device! As a result, the bitmap has the format you specify, but you can select it only into device contexts that are compatible with the bitmap format. For example, if you create a 4bpp bitmap, then you can select it only into 4bpp device contexts.[1]

Next up is the misleadingly-named `CreateDIBitmap` function. Even though the *DI* stands for device independent, this function does not create a device-independent bitmap. It creates a device-dependent bitmap that is compatible with the provided device context. The reason it's called *DI* is that you can provide pixels in a device-independent format, and those pixels will be used to initialize the bitmap. As noted in the documentation, the behavior is functionally equivalent to `CreateCompatibleBitmap` followed by `SetDIBits`.

If it's a device-independent bitmap you want, then the function to use is `CreateDIBSection`. The simplest use of this function creates a device-independent bitmap and gives you a pointer to the in-memory pixel buffer. You can then manipulate the pixel buffer directly, say, by `memcpy` ing the bytes from your original buffer.

The fancier use of this function creates a device-independent bitmap *around existing memory*. The memory needs to be in a file mapping object, either a file mapping object created from a file or (more often) a file mapping object created from the page file (in other words, a shared memory block). You can then specify the byte offset within the file mapping at which the pixel buffer starts. In this case, the memory is not copied; the memory in the file mapping object is the backing memory for the bitmap. If you modify the bitmap, then the contents of the file mapping object change; if you modify the contents of the file mapping object, you modify the bitmap.

Here's the table:

Function	**Type of bitmap**	**Resulting format**	**Source pixels**	**Must format match?**
`CreateBitmap`	Device-dependent	As specified	Copied	Yes
`CreateDIBitmap`	Device-dependent	Device-compatible	Copied	No
`CreateDIBSection` without `hSection`	Device-independent	As specified	Uninitialized (copy them yourself)	Yes
`CreateDIBSection` with `hSection`	Device-independent	As specified	Shared	Yes

In the above table, the *Resulting format* column describes the pixel format of the returned bitmap. The *Source pixels* column describes what happens to the pixels you pass as the source pixels: Are they copied into the bitmap, or does the bitmap share the memory with the source pixels? The *Must format match?* column specifies whether the format of the source pixels must match the pixel format of the returned bitmap. If *Must format match?* is *No*, then the system will perform a format conversion.

[1] Monochrome bitmaps are compatible with any device context and have special behavior when selected into color device contexts.

Raymond Chen

Follow