What is the difference between HINSTANCE and HMODULE?

devblogs.microsoft.com/oldnewthing/20040614-00

June 14, 2004

Raymond Chen

They mean the same thing today, but at one time they were quite different. It all comes from 16-bit Windows. In those days, a "module" represented a file on disk that had been loaded into memory, and the module "handle" was a handle to a data structure that described the parts of the file, where they come from, and where they had been loaded into memory (if at all). On the other hand an "instance" represented a "set of variables". One analogy that might (or might not) make sense is that a "module" is like the code for a C++ class – it describes how to construct an object, it implements the methods, it describes how the objects of the class behave. On the other hand, an "instance" is like a C++ object that belongs to that class – it describes the state of a particular instance of that object. In C# terms, a "module" is like a "type" and an instance is like an "object". (Except that modules don't have things like "static members", but it was a weak analogy anyway.) Here's a diagram. (Recall that we discussed 16-bit HRSRC in a previous entry.)

USER32 HMODULE			USER32 HINSTANCE
code segment descriptor	→	USER32 code…	USER32 data…
code segment descriptor	(not in memory)		
code segment descriptor	→	USER32 code…	
data segment descriptor			
HRSRC	(not in memory)		
HRSRC	→	USER32 resource…	
HRSRC	(not in memory)		
exports table			

In 16-bit Windows, all programs ran in a single address space, and if a DLL was used by five programs, it was loaded only once into memory. In particular, it got only one copy of its data segment. (In C++/C# terms, a DLL is like a "singleton class".) That's right, DLLs were system-global rather than per-process. The DLL did not get a separate copy of its data for each process that loaded it. If that was important to your DLL, you had to keep track of it yourself. In geek terms, there was only one "instance" of a DLL in the system. On the other hand, if you ran two copies of Notepad, each one got its separate set of variables – there were two "instances".

NOTEPAD HMODULE		HINSTANCE
code segment descriptor	→ NOTEPAD code…	NOTEPAD data…
code segment descriptor	(not in memory)	
data segment descriptor		HINSTANCE
HRSRC	(not in memory)	NOTEPAD data…
HRSRC	→ NOTEPAD resource…	

Both running copies of Notepad shared the NOTEPAD module (so the code and resources were shared), but each had its own copy of its variables (separate data segment). There were two "instances" of Notepad. The "instance" handles in the above diagrams are the data segments. Programs are identified by their the instance handle. You can't use the module handle, because the two copies of Notepad have the same module handle (since the same code is running in each). The thing that makes them different is that each has its own set of global variables. This is why the WinExec and ShellExecute functions return HINSTANCE: They are holdovers from 16-bit Windows, where HINSTANCEs were the way to identify running programs. The method by which code receives its HINSTANCE (i.e., knows where its global variables are) I will leave for a future article. It is somehow related to the now-obsolete MakeProcInstance function. When it came to design Win32, the question then arose, "What do we do with HINSTANCE and HMODULE for Win32?" Since programs ran in separate address spaces, you didn't have instance handles visible across process boundaries. So the designers took the only thing they had: The base address of the module. This was analogous to the HMODULE, since the file header describes the contents of the file and its structure. And it was also analogous to the HINSTANCE, since the data was kept in the data segment, which was mapped into the process directly. So in Win32, HINSTANCE and HMODULE are both just the base address of the module.

Tomorrow, I'll talk about that mysterious hinstPrev parameter to WinMain.

Raymond Chen

Follow