# Sidebox: Microsoft libraries

Modules are most obvious at link time: the application in example one must be linked with the 3 user libraries and then system libraries. This is where a particularly ugly linking problem can occur in Visual C++.

```
LINK : warning LNK4098: defaultlib "LIBCD" conflicts with use of other libs;
use /NODEFAULTLIB:library
```

What this error really means is something like: one part of the system was compiled to use a single threaded standard (libc) library with debug information (libcd) which is statically linked, while another part of the system was compiled to use a multi-threaded standard library without debug information which resides in a DLL and uses dynamic linking, or some something similar.

In fact VC++ comes with no fewer than six versions of the standard library: debug and non-debug, single-threaded and multi-threaded, .lib resident and .dll resident – there is no single threaded DLL version of the libraries.

Once this error appears people normally check their project settings run-time library choice. However, this is only part of the story and can actually be the start of many frustrating hours. Microsoft allow developers to imbed directives (#pragma comment(lib,....) ) in their own libraries which tell the linker which standard library this library expects to be linked with. These directives are all equal in the linkers eyes, so if your application is built for debug mutli-thread DLL and one of the libraries is set for single threaded debug you will have a conflict. Even though the project settings clearly say one thing the linker will try to also link an alternative.

The best way to fix this problem is to trawl through all your libraries and ensure they have the correct link settings. This can be very time consuming, particular as each one may need re-building. So, you have three options

? Ignore the warning, after all it is only a warning. However, your program now contains multiple instances of the same functions. If you are lucky you will quickly see an really obscure memory access violation, chances are the memory was allocated from one pool (say the libc, single threaded non-debug library) and freed to another (say, MSVCRTD, the multi-thread dynamic debug library). If you are unlucky things will work for you and a customer will encounter this problem.

? Use the linker option, /NODEFAULT:lib as suggested above, or even /FORCE. This is not a complete solution, even if you can get your program to link this way you are ignoring a warning sign: the code has been compiled for different environments, some of your code may be compiled for a single threaded model while other code is multi-threaded.

? Trawl through your libraries and ensure that everyone is set to the same settings.

However, even this is not guaranteed to work! There are two common problems here:

? You have a third party library which is linked differently to your application.

? You have other directives embedded in your code: normally this is the MFC. If any modules in your system link against MFC all your modules must nominally link against the same version of MFC. In reality this is not much of a bind, although your code may be set to link against MFC if the libraries will only be pulled in if something from them is used.

This can all be very boring and frustrating to track through your code. However, it's not that bad, there are a few tools you can use to look inside libraries.

? **dumpbin** supplied with VC++ has several options which can be helpful but /DIRECTIVES it probably the most useful.

? **depends** is a tool supplied with the NT resource kit can drill down into the DLL loaded by a program. (Actually, there are two versions of depends, a command line tool and a much nicer GUI one.)

? **pview** (supplied with both VC++ and the resouce kit can show which DLLs are loaded. However, it is quite normal to see MSVCRT and MSVCRTD (debug version) loaded at the same time as much Microsoft code itself uses MSVCRT.

In addition, if you follow these rules you should be OK:

? Decide which library you want to link against and stick with it for everything: normally this means dynamic multi-threaded debug and release. If you are using MFC you don't really have a choice.

? Ensure any third party libraries have this dependency (or better still none at all.)

? Never use /NODEFAULT:lib or /FORCE : once this problem enters your quickly ripples through everything, you can't allow exceptions.

? Never select "ignore default libraries" : it doesn't work very well anyway.

? Never use #pragma comment(lib,....) yourself. It introduces a hidden dependency.

Finally, the best solution as ever is to understand what is happening. Understand what libraries Microsoft supplies and is forcing on you.