# What are the porting issues going from VC8 (VS2005) to VC9 (VS2008)?

Asked 16 years, 1 month ago Modified 12 years, 9 months ago

Viewed 2k times

3

I have inherited a very large and complex project (actually, a 'solution' consisting of 119 'projects', most of which are DLLs) that was built and tested under VC8 (VS2005), and I have the task of porting it to VC9 (VS2008).

The porting process I used was:

1. Copy the VC8 .sln file and rename it to a VC9 .sln file.
2. Copy all of the VC8 project files, and rename them to VC9 project files.
3. Edit all of the VC9 project files, s/vc8/vc9.
4. Edit the VC9 .sln, s/vc8/vc9/
5. Load the VC9 .sln with VS2008, and let the IDE 'convert' all of the project files.
6. Fix compiler and linker errors until I got a good build.

So far, I have run into the following issues in that last step.

1) A change in the way decorated names are calculated, causing truncation of the names.

This is more than just a warning (http://msdn.microsoft.com/en-us/library/074af4b6.aspx). Libraries built with this warning will not link with other modules. Applying the solution given in MSDN was non-trivial, but doable. I addressed this problem separately in How do I increase the allowed decorated name length in VC9 (MSVC 2008)?

2) A change that does not allow the assignment of zero to an iterator. This is per the spec, and it was fairly easy to find and fix these previously-allowed coding errors. Instead of assignment of zero to an iterator, use the value end().

3) for-loop scope is now per the ANSI standard. Another easy-to-fix problem.

4) More space required for pre-compiled headers. In some cases a LOT more space was required. I ended up using /Zm999 to provide the maximum PCH space. If PCH memory usage gets bumped up again, I assume that I will have to forgo PCH altogether, and just endure the increase in what is already a very long build time.

5) A change in requirements for copy ctors and default dtors. It appears that in template classes, under certain conditions that I haven't quite figured out yet, the compiler no longer generates a default ctor or a default dtor. I suspect this is a bug in VC9, but there may be something

else that I'm doing wrong. If so, I'd sure like to know what it is.

6) The GUIDs in the sln and vcproj files were not changed. This does not appear to impact the build in any way that I can detect, but it is worrisome nevertheless.

Note that despite all of these issues, the project built, ran, and passed extensive QA testing under VC8. I have also back-ported all of the changes to the VC8 projects, where they still build and run just as happily as they did before (using VS2005/VC8). So, all of my changes required for a VC9 build at least appear to be backward-compatible, although the regression testing is still underway.

Now for the really hard problem: I have run into a difference in the startup sequence between VC8 and VC9 projects. The program uses a small-object allocator modeled after Loki, in Andrei Alexandrescu's Book *Modern C++ Design*. This allocator is initialized using a global variable defined in the main program module.

Under VC8, this global variable is constructed at the very beginning of the program startup, from code in a module crtexe.c. Under VC9, the first module that executes is crtdll.c, which indicates that the startup sequence has been changed. The DLLs that are starting up appear to be confusing the small-object allocator by allocating and deallocating memory before the global object can initialize the statistics, which leads to some spurious diagnostics. The operation of the program does not

appear to be materially affected, but the QA folks will not allow the spurious diagnostics to get past them.

Is there some way to force the construction of a global object prior to loading DLLs?

What other porting issues am I likely to encounter?

visual-c++ visual-c++-2008 name-decoration vc8

Share

Improve this question

Follow

edited May 23, 2017 at 12:08

community wiki
6 revs, 3 users 64%
Howard Lee Harkness

## 5 Answers

Sorted by: Highest score (default)

1

> Is there some way to force the construction of a global object prior to loading DLLs?

How about the DELAYLOAD option? So that DLLs aren't loaded until their first call?

Share Improve this answer

Follow

answered Feb 3, 2009 at 20:22

community wiki
Die in Sente

---

1

That is a tough problem, mostly because you've inherited a design that's inherently dangerous because you're not supposed to rely on the initialization order of global variables.

It sounds like something you could try to work around by replacing the global variable with a singleton that other functions retrieve by calling a global function or method that returns a pointer to the singleton object. If the object exists at the time of the call, the function returns a pointer to it. Otherwise, it allocates a new one and returns a pointer to the newly allocated object.

The problem, of course, is that I can't think of a singleton implementation that would avoid the problem you're describing. Maybe this discussion would be useful: http://www.oneunified.net/blog/Personal/SoftwareDevelopment/CPP/Singleton.article

Share Improve this answer Follow

edited Feb 4, 2009 at 20:51

community wiki
2 revs
Ori Pessach

Turned out that using singletons was the best answer. Thanks! – Howard Lee Harkness Aug 7, 2014 at 16:22

0

That's certainly an interesting problem. I don't have a solution other than perhaps to change the design so that there is no dependence on undefined behavior of the order or link/dll startup. Have you considered linking with the older linker? (or whatever the VS.NET term is)

Because the behavior of your variable and allocator relied on some (unknown at the time) arbitrary order of startup I would probably fix that so that it is not an issue in the future. I guess you are really asking if anyone knows how to do some voodoo in VC9 to make the problem disappear. I am interested in hearing it as well.

Share Improve this answer Follow

answered Oct 31, 2008 at 20:30

community wiki
Tim

0

How about this,

1. Make your main program a DLL too, call it main.dll, linked to all the other ones, and export the main function as say, mainEntry(). Remove the global variable.

2. Create a new main exe which has the global variable and its initialization, but doesn't link statically to any of the other application DLLs (except for the allocator stuff).
3. This new main.exe then dynamically loads the main.dll using LoadLibrary(), then uses GetProcAddress to call mainEntry().

Share Improve this answer Follow

answered Feb 3, 2009 at 23:37

community wiki
Carlos A. Ibarra

---

0

The solution to the problem turned out to be more straightforward than I originally thought. The initialization order problem was caused by the existence of several global variables of types derived from std container types (a basic design flaw that predated my position with that company). The solution was to replace all such globals with singletons. There were about 100 of them.

Once this was done, the initialization (and destruction) order was under programmer control.

Share Improve this answer Follow

answered Mar 28, 2012 at 20:07

community wiki
Howard Lee Harkness