DCMTK - Bug #408

Warnings with VisualStudio (Windows) when building DLLs

2012-05-03 17:22 - Jörg Riesmeier

Status:	Closed	**Start date:**	2012-05-03
Priority:	Normal	**Due date:**	
Assignee:	Jan Schlamelcher	**% Done:**	50%
Category:	Library	**Estimated time:**	0:00 hour
Target version:	3.6.2		
Module:	dcmsr	**Compiler:**	
Operating System:			

Description

There are various warnings of the following type:

```
C:\Source\dcmtk-3.6.1_20120502\dcmsr\include\dcmtk/dcmsr/dsrtlist.h(263) : warning C4661: 'const D
SRGraphicDataItem DSRListOfItems<T>::EmptyItem' : no suitable definition provided for explicit tem
plate instantiation request
        with
        [
            T=DSRGraphicDataItem
        ]
        C:\Source\dcmtk-3.6.1_20120502\dcmsr\include\dcmtk/dcmsr/dsrtlist.h(219) : see declaration
 of 'DSRListOfItems<T>::EmptyItem'
        with
        [
            T=DSRGraphicDataItem
        ]
```

Related issues:

Related to DCMTK - Bug #694: Problems (again) with "dcmsr: EmptyItem" and bui...	**Closed**	**2016-09-20**

History

#1 - 2012-11-13 11:24 - Uli Schlachter

Google gave me:

http://social.msdn.microsoft.com/Forums/en-US/vclanguage/thread/1cdfb076-ca3c-4469-a4fd-48201ac97714

This boils down to "templates are weird". Apparently, we are supposed to do the following in a **header** in global context:

```
template<typename T> const T DSRListOfItems<T>::EmptyItem;
```

In some cpp-file, we would then do template specialization magic:

```
template<> const DSRGraphicDataItem DSRListOfItems<DSRGraphicDataItem>::EmptyItem(0, 0);
```

Also, we might need explicit template instantiation:

```
template class DSRListOfItems<DSRGRaphicDataItem>;
```

I haven't checked, but somehow I feel like MSC6 wouldn't like this solution...

#2 - 2012-11-19 17:48 - Jan Schlamelcher

- File 0001-trial-patch-to-fix-Bug-408.patch added

- % Done changed from 0 to 50

Solved the issue using Meyers Singleton instead of static members (see patch proposal)
Advantages:
- No need for explicit instatiations in child classes (fixes the error)
- Foolproof for using dlls (no danger of multiple instatiations of the same static member being present)
- Automatically threadsafe when using a c++11 compiler
Disadvantages:
- Restricts the way items can be constructed as it relies on the constructor being called with a single integer parameter of value zero to construct the

default item (currently the case for every subclass anyway)

Possible fix for disadvantage (if required):
Option 1 - Rely on type traits: Create type-traits classes to distinguish POD- and non-POD types and either call the default constructor or initialize it to zero (would still restrict elements in the way the default constructor without parameters would be required to construct the default item, but would be a cleaner solution than the current one).
Option 2 - Use CRTP to let each child class implement their own defaultElement() method (maximum flexibility but possibly unnecessary amout of code required for each child class; more complex and less readable)
Option 3 - Default element construction policy with the current behaviour as default policy (impossible for some compilers)

#3 - 2012-11-20 10:11 - Uli Schlachter

> Restricts the way items can be constructed as it relies on the constructor being called with a single integer parameter of value zero to construct the default item (currently the case for every subclass anyway)

Well, I guess that works only accidentally for DSRGraphicData3DItem, but yeah, no problem here.

However, the missing thread-safety might be a problem (It's 2012 and we are still using compilers from 1998, so C++11 doesn't help us much here) and I don't see how this problem could easily be solved.

#4 - 2012-11-20 12:46 - Jan Schlamelcher

Afaik the previous approach wasnt thread-safe either, so i listed thread-safety under advantages since the new approach would at least be threadsafe under c++11. Also, if you call emptyItem() for the first time when still in a single threaded environment, it is threadsafe for the rest of the execution time. This could be said in the documentation like: If you want to use the dsr module in a multi threaded environment, ensure to call emptyItem() at least once before launching the other threads (when not using c++11)

#5 - 2012-11-20 13:02 - Uli Schlachter

The previous approach called the constructor as a static initializer of a global variable, thus before main() was called. Anyone who starts threads before main() runs deserves what he gets. ;-)
(Oh, I wonder what happens / happened when libdcmsr.so / libdcmsr.dll is loaded via e.g. dlopen(), then dlclose()d again and dlopen()d again...)

Anyway, unless someone comes up with a better approach, we now have to decide if we want the patch or not. I guess I vote for the patch, because the warnings are more annoying than a race which **hopefully** doesn't do anything bad (Constructing an int twice doesn't do anything bad and I think the non-POD types don't do anything bad if constructed twice either, because they only set some member variables).

#6 - 2012-11-21 13:20 - Jan Schlamelcher

- File 0002-thread-safeish-dcmsr-behaviour-for-ancient-compilers.patch added

Uli Schlachter wrote:

> The previous approach called the constructor as a static initializer of a global variable, thus before main() was called. Anyone who starts threads before main() runs deserves what he gets. ;-)

I see, didn't think about that small difference. In this case i would go for the following hybrid approach (see patch-patch).
As far as i understand it, this would combine the advantages of both strategies, without their disadvantages.
And it also works for vc6...

#7 - 2013-09-12 20:44 - Jörg Riesmeier

- Category set to Library

#8 - 2017-03-24 13:52 - Jan Schlamelcher

- Status changed from New to Closed

- Assignee set to Jan Schlamelcher

Closed by commit #97d6fee689ab94f26

Files

0001-trial-patch-to-fix-Bug-408.patch	5.85 KB	2012-11-19	Jan Schlamelcher
0002-thread-safeish-dcmsr-behaviour-for-ancient-compilers.patch	1.18 KB	2012-11-21	Jan Schlamelcher