DCMTK - Feature #130

support for Metrowerks CodeWarrior compiler on the Mac

1998-11-19 00:00 - Marco Eichelberg

Status:	Rejected	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0:00 hour
Target version:			
Module:	dcmdata	**Compiler:**	
Operating System:			

Description

reported by Arnaud Masson

I have made these changes because with Metrowerks CodeWarrior compiler on
the Mac, static object may be not well initialized in static libraries.
Anyway I think it's better to initialize only global pointers to NULL and
use the "lazy initialization" pattern, ie build globals objects only the
first time they are needed and always access them only through accessor
functions, not through global variables.

==== dcdebug.cc ====

```
***Original:
```

```
FILE * DcmDebugDevice = stdout;
```

```
***Modified:
```

```
FILE* gDcmDebugDevice = NULL;
```

```
FILE* Get_DcmDebugDevice() {
```

```
if (gDcmDebugDevice == NULL)
gDcmDebugDevice = stdout;
```

```
return gDcmDebugDevice;
}
```

==== dcdebug.h ====

```
***Original:
```

```
extern FILE * DcmDebugDevice;
```

```
***Modified:
```

```
extern FILE* Get_DcmDebugDevice();
#define DcmDebugDevice Get_DcmDebugDevice()
```

==== dcdict.cc ====

(1)

```
***Original:
```

```
DcmDataDictionary dcmDataDict(OFTrue, OFTrue);
```

```
***Modified:
```

```
static DcmDataDictionary* gDcmDataDictionary = NULL;

DcmDataDictionary& Get_dcmDataDict() {

if (gDcmDataDictionary == NULL) {
gDcmDataDictionary = new DcmDataDictionary(OFTrue, OFTrue);
}

return *gDcmDataDictionary; // warning: reference !
}
```

(2)

```
***Original:

DcmDataDictionary::DcmDataDictionary(OFBool loadBuiltin, OFBool
loadExternal)
{
   loadSkeletonDictionary();
...
}

***Modified:

DcmDataDictionary::DcmDataDictionary(OFBool loadBuiltin, OFBool
loadExternal)
{
skeletonCount = 0; // <--- new
dictionaryLoaded = OFFalse; // <--- new

   loadSkeletonDictionary();
...
}
```

==== dcdict.h ====

```
***Original:

extern DcmDataDictionary dcmDataDict;

***Modified:

extern DcmDataDictionary& Get_dcmDataDict();
#define dcmDataDict Get_dcmDataDict()
```

Hello Marco

> I will integrate your changes into the next release of the DCMTK,
> but probably in the context of a compile time flag, i.e.
> #ifdef NO_STATIC_CTORS_IN_LIBS
> <your code>
> #else
> <existing code>
> #endif

Thanks!

> I see two problems with your modifications:
> 1. In the way it is currently implemented, the destructor for the
> DICOM dictionary is never executed. While this causes no harm with
> the current implementation, it is at least annoying when you attempt
> to find memory holes (because there will always be some 3000+ "holes"
> reported for the dictionary) and it might cause harm should a future
> version of the dictionary ever require its dtor to be executed

in order to free certain ressources (say, temporary files, semaphores,
whatever).

This can be solved by creating a small function that deletes
the dictionary and is registered as an atexit() function.
This is probably the way I will implement it in the next DCMTK release

Why don't you use a pair of functions like

```
void InitDCMTKLibrary();
  void CleanupDCMTKLibrary();
```

Many libraries use this pattern (OLE, QuickTime...).
These functions may have an internal counter so that the InitXXX can be
called several times but each call must be balanced with a call to
CleanupXXXX.

2. In future versions of the toolkit there might be code that requires
the ability to initialize static objects from libraries before main().
In principle, this is a very convenient way for libraries to register
functions before main() gets executed.
For instance, we are thinking of a mechanism for adding decompression
functions (for compressed images) that work with all DCMTK applications
reading image data without any need to change these applications.
Just link the application with a "decompression library" that tells
DCMTK that decompression code for a certain transfer syntax is available
before main() gets executed.
You simply cannot mimick this behaviour without modifying many
main()s..

Yes, it is one of the best things with static objects.

However there is also another problem with them on the Mac.
By default, the CodeWarrior Mac PowerPC Linker (unlike x86 linkers) sees
static object classes that are NEVER referenced as dead code. This is an
useful optimization. Of course the dictionary is referenced from other
files, so it works.
But auto-registering objects don't work since the goal is to never
reference them directly.
This behavior can be changed but if the option is off, dead code that is
really unused is linked in the final exe.

Maybe you could put the auto registration static objects between
preprocessor directives #ifndef NO_STATIC_CTORS_IN_LIBS - #endif and allow
to use the old method to register from main.

You could also put decompressors in separate DLLs in a predefined directory.
At startup, the DCMTK lib would scan the directory and load all
decompressors.
Each DLL would export a function DLLGetDCMTKDecompressor() returning an
instance of the decompressor, like COM.

I think that's an useful modification because
1) generally in C++ you can't control the order used to call static objects constructors,
2) as soon as you link the OFFIS code with an application, all global objects are created, which may not be a good thing if these objects require some explicit initialisations from the application,
3) a global accessor function is cleaner than a direct access to a global variable. Usually a static method is used, this is the "Singleton" of the book "Design Patterns".
...
So even if there is no problem on PC and UNIX yet, the lazy initialisation
pattern "construct only when needed" seems better.

History

#1 - 2017-04-20 17:57 - Marco Eichelberg

- Status changed from New to Rejected

This issue only affects an old compiler on an operating system (Mac OS classic) that is not supported anymore. Won't fix.