attacks if they had to confront their targets face-to-face? Or are we finally witnessing the effects of years of easy access to hardcore pornography on child surfers? Perhaps it's a combination of things, or something else altogether.

I don't know what's happening or why it's happening. The trend is insidious, but it's definitely there. I only know that I don't like it.

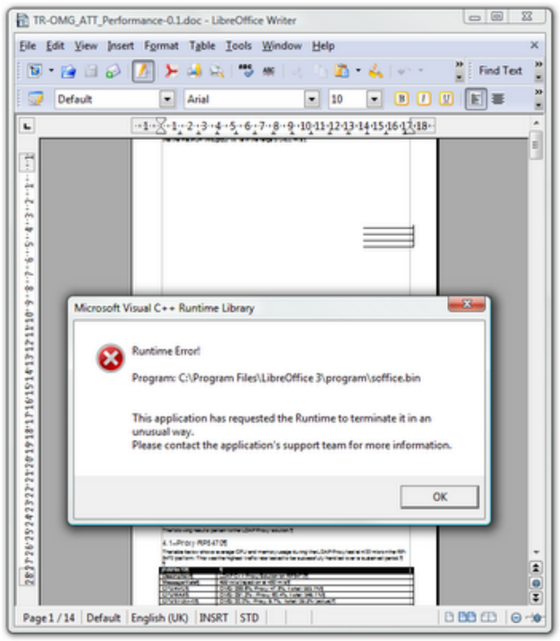
From the **Big Angry Blog** Copyright 2011 Andy Thomas

()

## **LibreOffice Fail/Crash** **4 Mar 2011, 9:13 pm**

I'm just about to uninstall LibreOffice and revert to OpenOffice.

Here's why...


TR-OMG_ATT_Performance-0.1.doc - LibreOffice Writer
File Edit View Insert Format Table Tools Window Help
Find Text
Default
Arial
10
OK
Page 1 / 14 Default English (UK) INSRT STD

Microsoft Visual C++ Runtime Library

Runtime Error!

Program: C:\Program Files\LibreOffice 3\program\soffice.bin

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.

Apart from the frequent crashes, every time LibreOffice starts up it seems to presents me with the "document recovery" window.

Just wondering whether others are experiencing similar problems. (I'm running Windows if it's not obvious from the screenie.)

I do appreciate the effort the guys at LibreOffice have put in, and hope they get things right in the next release.

From the **Big Angry Blog** Copyright 2011 Andy Thomas

()

## **Building Nokia QT SDK and Making Phonon Work** **2 Feb 2011, 6:18 am**

This article is, in part, about QT, C++ programming and getting the damn Phonon multimedia library to work on Windows. But first a little background...

**Back in 2004**, I decided to write a quick game. It was my first since the halcyon days of the 1980s, and it occurred to me that, perhaps, the world needed another Asteroid re-make. But this wasn't just about writing an Asteroids clone. I wanted to try a different way of writing applications.

My chosen development environment at the time was C++ Builder on Windows. I loved the Borland VCL development framework, having used Delphi years earlier, but realized that it was time to move on. With my Asteroid remake, I wanted to write as much back-end code using only the vanilla C++ Standard Library, and keep the GUI as a light-weight wrapper around it. This would be mean that I wouldn't be tied to any one GUI development framework as I could, without too much effort, swap out the front-end GUI code for another.

I used Borland C++ Builder to get something working initially, and remember being quite shocked that it did! As I had written all the asteroid modelling and collision code from scratch, based loosely on conservation of momentum laws, I was kind of expecting to spend a few weeks debugging it before anything would render. Fortunately it worked more or less out of the box, so I was able to turn my attention to what I really wanted to do, which was to use my little project as a vehicle to develop cross-platform code. So I swapped out the Borland front-end stuff and replaced it with wxWidgets, an open source development framework. I initially released this wxWidgets version of the game under the name of "Celesteroids".

A few years later and I was experimenting with Nokia's QT Framework, and decided to re-use my Asteroid game as a test-bed. I swapped out the wxWidgets front-end for QT, and took the opportunity to rename the application "Asteroid Cresta" (because I thought "Celesteroids" sounded a bit rubbish, but couldn't think of anything better at the time).

Here's some game play of Asteroid Cresta...

It's available under GPL and you can download it here.

If you're wondering when I'm going to start talking about Phonon, I'm coming to it...

One of the issues I had with QT was audio. The wxWidgets Celesteroids version of the game had fabulous music and explosion sound effects. QT did have a multimedia API called Phonon, but unfortunately, this could only be used on Windows with Microsoft Visual C++ and not MinGW. With MinGW, I was stuck with the QSound class which, on Windows at least, appeared to be nothing more than a wrapper for the PlaySound() Win32 API function. Consequently, the initial version of Asteroid Cresta had very limited sound which was a bit disappointing.

Some time later, when I had little more free time, I rebuilt Asteroid Cresta with Visual C++ and got the music and sounds to work using Phonon. I hurriedly packaged this up into the installer and released the new version. Sadly, as I soon discovered, the audio didn't seem to work on computers without an installation of the QT development SDK, and at the time, I was already sucked into other projects so it didn't get fixed until recently.

Over the last few months I've been developing with QT in anger and I absolutely love it. My focus at the moment is Windows, but I also develop on Linux and love the cross-platform capability that QT offers. I've spent a lot of time recently building the QT framework for the both MinGW and MSVC, and unfortunately, I find the time and effort (but mostly the time) required to do this to be one of the few downsides of QT and a particular bone of mine. The QT SDK is pretty large and takes an eternity to simply unpack and copy around your hard-drive, not to mention, to compile it. I've wasted a lot days building it only to find that I had used the wrong static library setting or something, or had built it in the wrong location on my hard-disk, and had to start all over again. As QT takes several months to build (OK I exagerate, but half a day on my system) this was not something I enjoyed.

So I thought I'd share some of the experience and caveats I've picked recently, starting with how to deploy Phonon. **Note** that my particular development environment comprised QT 4.7.1, MinGW with GCC 4.5, and MS Visual C++ 2010.

**Deploying Phonon (phonon_ds94.dll)**
The reason why my poor Asteroid Cresta game stayed silent when ran other machines was the fact that I had omitted to deploy the "Phonon back-end DLL". Well, I didn't omit it really, I just didn't know about it--I just love how they don't tell you **really really important** stuff like this (well actually they did, but it was buried in an FAQ).

When deploying your Phonon based application, you need to create a sub-directory called "**phonon_backend**" in the main application directory. In there, you need to deploy **phonon_ds94.dll** which can be found in the "plugins" sub-directory of your QT deployment once you have built it.

Additionally, as you probably know, you will need to deploy **phonon4.dll** in your application directory along with the other QT DLLs.

**Deploying QT Applications Generally (Required DLLs)**
Assuming that you are using the dynamically linked build of QT, then there are a number of DLLs you will need to deploy. The two core ones you probably already know of are: **QtCode4.dll** and **QtGui4.dll**.

Others are needed according to which modules you have linked to. Before releasing any application, I strongly suggest that you run Dependency Walker against your executable to determine which DLLs it requires.

If you are building with Visual C++, then you will almost certainly need the C and C++ runtime libraries also. These are the following re-distributable DLLs shipped with MSVC:

**msvcrXXX.dll**
**msvcpXXX.dll**

I located these DLLs in my Windows System32 directory. Note that "XXX" defines the compiler version used, and with VC 2010 the version number is 100, i.e. "msvcr100.dll". For VC 2008, it is 90.

Now, in principle, if you build your application with the "/MT" (static C runtime) option, as opposed to the "/MD" (dynamic runtime) option, then you will not need to deploy the above C/C++ runtime DLLs. In this case, however, you will need also to build the entire QT SDK using the MT static C runtime option (more below), and any other third-party libs you are using. This can prove problematic and you will find yourself with linker errors if you have static libraries built with mixed MT/MD options. If you download the ready built version of QT, it will be built with the "/MD" option.

I have a preference for applications with minimal DLL dependencies, but in this case with MSVC, I eventually gave up and just accepted that I would build my applications with the "/MD" option.

Applications built with MinGW link to the **msvcrt.dll**, which is available in the System32 directory on all Windows systems as standard. There is no need to ship C runtime libraries in this case, although you may need the infamous MinGW **mingwm10.dll** file, although Dependency Walker will tell you if you do or not.

**Multiple Compiler and QT Version Deployments**
It's possible to have multiple QT SDK builds on your system for use with different compilers. However, I would recommend that you plan on where you are going to locate these before building QT as it's very time consuming to start over again (and again).

**IMPORTANT:** The directory where you build the QT SDK matters -- it is fixed at compile time and you can't relocate it afterwards. A mistake I have made numerous times is to unpack the SDK in a temporary location and build it there, and then attempt to move it to a final directory afterwards. This does not work and **qmake** will give you errors when you come to use it.

You need to decide where you are going to put the SDK before you configure and build, and personally I chose a directory naming convention like this:

4.7.1-mgw45
4.7.1-vc2010

I version number both the QT SDK and the compiler used to build it, as you cannot link an application built with GCC 4.5 against an SDK built with GCC4.4. The same applies to Visual C++.

Furthermore, the QT installation documentation tells you to create an environment variable called "**%QTDIR%**" and add the QT bin directory to your path. This appears to be a suggestion only, which is fortunate if we have multiple installations. Personally, I create multiple variables, such as "%QT_MGW%" and "%QT_MVC%" and use these to explicitly call the correct build of qmake.exe when compiling my project.

**Configure and Build QT with MinGW**
After many time consuming iterations, I landed on a particular QT MinGW build configuration which works for me and gives me the stuff I need, but cuts out the guff I don't. And here it is:

configure **-release** -shared **-no-exceptions** -platform win32-g++ -no-qt3support -no-libtiff -no-libmng -no-audio-backend -no-phonon -no-phonon-backend -no-multimedia **-no-style-motif -no-style-cde -no-style-plastique -no-script -no-scripttools -nomake examples -nomake demos**

This builds the shared (DLL) version of QT without Phonon (which MinGW doesn't support anyway). I've highlighted a few important choices:

**-release** : It takes twice as long to build both the release and debug versions of QT. I personally don't use the debug version, so why build it? You may want to do so however.

**-no-exceptions** : I found this to be valuable! More about this below.

**Others** : Why waist time building examples and demos, and a non-native "look and feel"?

With regard to the "-no-exceptions" option, with GCC 4.5 the size of resulting DLLs increased massively and QT DLLs built with this compiler tend to be huge. I found that by building without exceptions (which QT doesn't use), the size of the resulting QT DLLs was reduced by about half.

You can still use exceptions in your own applications, but if you use "-no-exceptions" in building QT itself, you will need to explicitly configure them in your application's qmake pro file.

You will need to add the following to your CONFIG statement:

**CONFIG += exceptions**

Failure to add this will mean that your applications won't build.

**Configure and Build QT with Visual C++**
Here's my configure command for Visual C++ 2010:

configure -release -shared **-exceptions** -platform **win32-msvc2010** -no-qt3support -no-libtiff -no-libmng -audio-backend -phonon -phonon-backend -multimedia -no-style-motif -no-style-cde -no-style-plastique -no-script -no-scripttools -nomake examples -nomake demos

Unlike the MinGW configure, this will build the Phonon libraries.

You can change the **win32-msvc2010** option to suit your compiler version. Note that I am building with "-exceptions" in this case because I found that it didn't seem to make a huge difference to the output DLL size with Visual C++.

**Miscellaneous Caveats**

**QT designer crashes with MinGW** - I don't know whether anyone else found this, but I discovered that the MinGW build of designer.exe crashes intermittently, whereas the Visual C++ build is stable. I took a look at it with Dependency Walker and found that it was missing the **IeShims.dll** library. I manage to locate this in the "%ProgramFiles%\Internet Explorer" directory and copied it across into QT's bin directory. The designer application has been stable since.

**The MinGW build fails with qjpeghandler.cpp** - This applies to QT version 4.7.1. After a bit of searching, I found the following simple fix.

In the file "src/gui/image/qjpeghandler.cpp", change the line:

**#if defined(__RPCNDR_H__) && !defined(boolean)**
typedef unsigned char boolean;
define HAVE_BOOLEAN
...

to

#if defined(__RPCNDR_H__) **||** !defined(boolean)

and just re-make to carry on building (no need to re-configure).

**Building the static C runtime QT with Visual C++** - It is possible to build QT with the "/MT" static runtime library option, and in this case, your application won't need the **msvcrXXX.dll** and **msvcpXXX.dll** DLLs.

To do this, before you configure and build QT, change the applicable MSVC qmake.conf file in your mkspecs directory to:

QMAKE_CFLAGS_RELEASE = -O2 -MT

and:

QMAKE_LFLAGS_RELEASE = /INCREMENTAL:NO /NODEFAULTLIB:MSVCRT

However, I personally found that while I am able to build "/MT" applications without problems, QT Designer crashes when built with it. This was one of the reasons I settled on using "/MD".

**A Final Thought...**
All in all, we C++ developers may feel proud that we are not forced to use a more productive development technology such as Java. (Or at least that we are better programmers any way!)

From the **Big Angry Blog** Copyright 2011 Andy Thomas

## Retro with DreamCalc 1 Jan 2011, 7:09 pm

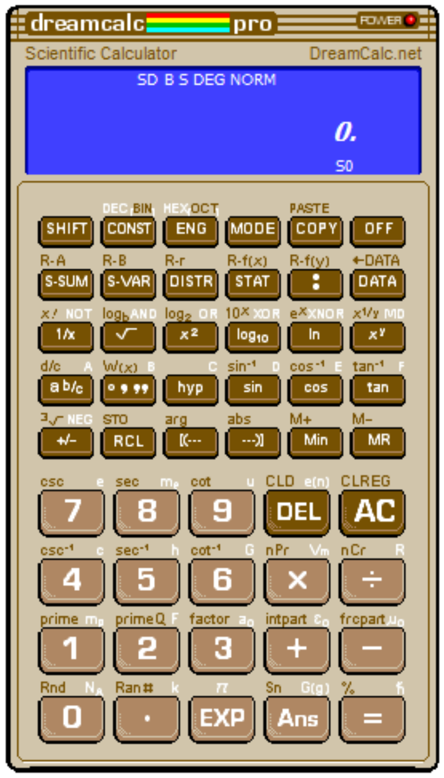
I've just put out the latest release of DreamCalc, version 4.7.1. (Users can get it here.)

While updating the software, I decided to have a look back at DreamCalc in the early days of 2004. I had a moment of silliness back then, and DreamCalc version 2.1.3 had some unusual *retro computing skins*.

For posterity, I grabbed some screen dumps, and here they are below. They were removed in subsequent releases of DreamCalc.

The history of DreamCalc actually stretches back to 1996, and here is also a brief write up of how the software developed.

Here's DreamCalc disguised as a Commadore 64....



And here it is as the Sinclair ZX Spectrum.