Note 169: qmake Quick Tutorial

Using qmake and how to deal with errors

Kamran Tavakoli (DRAO)
A. G. Willis (DRAO)

1 Introduction

The script called `qmake` is a wrapper program for Aips++ gmake programs. Although gmake produces executables on all SUN installations it runs into problems when an IBM RISC 6000 is used. The C++ compiler used on the IBM machines is xlC, and is run under the AIX operating system. A linker helper version of xlC (tlC) is available at DRAO. tlC is a binary wrapper that is supposed to speed up the linking process. Essentially qmake uses the existing setup of Aips++ and needs minimal modification to site makedefs only.

2 qmake's FAQ

2.1 Why IBM cannot produce executables in reasonable time without qmake?

IBM's version of ptrepository is called tempinc. This directory declares the template instantiations, and when an executable is being produced, the compiler looks at this directory to know which template instances are used, and creates object modules out of them. The problem is that this area should only contain the instantiations that are related to the executable being produced. Anything extra will be taken as a part of the executables being produced. This causes unresolved symbols at worst and HUGE executable size and a very long bind time at best.

2.2 Does using qmake interfere with using gmake?

No, qmake uses gmake, and if gmake is used in the traditional way (by itself) it works as well. That is, sometimes after an error has occurred in a file, you may want to recompile that file to see if the error is gone, you may use gmake to do that. The only changes to the makefiles are made to site makedefs.

2.3 How does qmake avoid the problem?

qmake separates all concerns, that is, compilations of differently named files have nothing to do with each other. They have separate library and template repository (tempinc in IBM lingo). In doing so qmake avoids the annoying problem of unresolved symbols (unless it is a genuine one), and cuts down the compile and bind time. After qmake is run for the first time (using the -f flag), and the library and repository is filled, the amount of time to produce an executable after changing the source of that executable is minimal.

qmake has a few modes of operation but the first time that it is used to produce an executable it does the following:

```
qmake does this:
- calculate dependencies of the TestFile at hand (call qdeps)
     qdeps does this:
     - calculate and write the dependencies (.cc) to temporary files
     - compile .o files out of these dependencies and put in lib
     - make a private library in libqmake called "libaipsMyTestFile.a"
```

```
          - fill up "tempincTestFile" (unique to this exe) with instances
     - call gmake after setting up some exported shell variables
     - cleanup
```

It is worth noting that qmake creates one tempinc per executable, that is, the name of the executable is appended to "tempinc" to produce "tempincMyTestFile"; this directory resides under programmer's "aips++/ibmrs/". Also to make sure that gmake (GMAKE) does not mess up the library area, a new library area is used for qmake. This new directory is called "libqmake" and resides under "aips++/ibmrs".

The existence of a separate tempinc for each executable is absolutely necessary for the proper working of the system. But using libqmake, although not necessary all the time, avoids irritating link errors, that can be caused by interleaving gmake and "qmake -a MyTestFile".

When qmake is called for the first time (with the -f flag), "qmake" calls "qdeps" to calculate dependencies and build the private library; in doing so the compiler writes the necessary instantiation files to the the unique "tempincMyTestFile" area. Later when qmake calls gmake to make an executable, it compiles the executable's source and puts the object module in the programmer bin area (same as the original Aips++ gmake setup). Then the compiler builds the object modules for each instantiation file that reside in "tempincMyTestFile". Then the linker links these with the ".../libqmake/libaipsMyTesFile" and produces the executable that will be placed in the programmer bin area.

From here on, you use "-a" and "-n" flags mostly unless a complete and fresh rebuild of the library and executable is desired, which can be achieved using "-f".

2.4 What are the flags all about?

```
qmake -f myfile          // to flush and rebuild exe from scratch, no surprises
qmake -f myfile -O       // same as above but compile optimized
qmake -f myfile -g       // same as above but compile with debugging info

qmake -a myfile          // to append to (libqmake/tempinc) and rebuild exe
qmake -a myfile -O       // same as -a but optimized
qmake -a myfile -g       // same as -a but with debugging info

qmake -n myfile          // to rebuild exe, no dependency, no question
qmake -n myfile -O       // same as -n but optimized
qmake -n myfile -g       // same as -n but with debugging info

qmake myfile             // for interactive mode (adds nothing to the above!)
qmake myfile -O          // interactive, and Optimized
qmake myfile -g          // interactive, with debugging info

qmake -d myfile          // only produce a file containing dependencies
qmake -h                 // to get some help
```

The "-O" and "-g" flags are used to either optimize or add debugging info to the executable and the library as well as template instances. Use them as you need, but remember to use them consistently for best results. That is, if they are used the first time around they should be used for all compiles of that executable.

The "-f" flag is used the first time to "force clean" the library and tempinc related to that executable and re-do every thing from scratch. It does not have to be repeated unless building from scratch is desired.

The "-a" flag is used after "-f" is used at least once. It recalculates dependencies and appends any module that failed the first time to your library. Then it tries building the executable. So you use "-a" after a "-f" compile that encountered some error(S) in building the library or instantiating the templates, or if unresolved symbols where encountered. Make sure the source of the error is corrected before doing a "qmake -a MyTestFile".

The "-n" flag is used after "-f" is used at least once and an executable is produced; "-n" stands for "no dependency calculation", thus gmake is called directly after the internal varibles are set up. The "-n" compile is quick and should be used every time you modify the source file (MyTestFile.cc).

The "-d" flag is used to create a text file of dependencies. It does nothing to create or change the library or call gmake. The "-h" will provide some help. The flags are designed to work as shown above, you cannot mix and match them in any other way.

2.5 What are the necessary changes to the system?

Since at this point the directory holding qmake and other related files is not determined, I could not instruct you as to which directory must be included in your path. Jim Horstkotte will determine that and this file will be revised to reflect that info. For now qmake and companion files must be in "some" directory that is included in the path, and that directory must be assigned to the variable SEDDIR which is found in the qmake and qdeps scripts.

First: create a subdirectory called "libqmake" under programmer's "aips++/ibmrs/".

Second: modify your site makedefs in the following way: [Note that $(QMAKE) and $(NAME), that are seen here are exported by the qmake script]

```
# ADD THESE DIRECTLY TO YOUR SITE'S "makedefs" FILE
# make sure they are placed at the right spot in your file
# before they are used by other statements.
=====================================================================
# to exclude the system's class library from the programmer link list
ifeq "$(MAKEMODE)" "programmer"
        LIBaips := defeat defeat defeat defeat static defeat
endif

# this line is needed for qmake to use libqmake instead of lib,
# regular use of gmake will not be affected
PGMRLIBD := $(wildcard $(PGMRARCH)/lib$(QMAKE))

# this line added so that the name of library is different when using qmake
# and the same when using gmake directly.
EXTRA_PGMRLIBS := $(EXTRA_PGMRLIBS) $(PGMRLIBD)/lib$(PACKAGE)$(NAME).a

# defining the tempinc for IBM
# the $NAME is used by qmake not gmake, so if qmake is used the tempinc
# will have the name such as tempincMyfile but if gmake is used directly
```

```
# then the name tempinc is used
# Make sure it is at the right spot in your file, and is used properly
# by C++FLAGS and other varibles.
PGMRPTRD := $(PGMRARCH)/tempinc$NAME

==========================================================================
```

and make sure wherever you refer to -qtempinc=... you have it in this format:

```
-qtempinc=$(PTRDBGD)
```

this is the most important part of the qmake, since the variable $NAME will differentiate between tempincs and libs.

3 Dealing With Errors

qmake goes through 2 different phases. First, it calculate dependencies (qdeps). You should not have any errors here, but if any errors are encountered they originate from the qdeps script and usually the use of "sed". Some of the older Aips++ files do not reflect the new subdirectories added to the Aips++ implement and include areas, and qdeps relies on this include information to calculate the dependencies. The first phase also compiles the dependencies and creates a library (qdeps). If there are syntax errors in any of the dependency files, your library will not include them, see bellow for solution.

Also, if new directories are introduced to include or implement areas, the script qdeps has to modified to accomodate for these directories. [To be exact these functions have to be modified: create_implement_subdirs(), make_o_files(), add_the_extra_o(), strip_subdirs(). To see what must be done, one should note the surrounding comments].

The Second phase is building the executable, which produces MyTestFile.o and links it to tempinc's ".o" files and the library. Errors can be reported here as well.

3.1 What to do if the compiler reports error(s) during building of library?

Errors at this stage do not stop the operation, so you can look back at your log file and find them. Then correct the error and test it in the implement area using regular "gmake MyFile.o". Once it is gone, use "qmake -a MyTestFile".

3.2 What to do if compiler reports error(s) during the template instantiation?

Trace the source of error (which file in tempincMyTestFile is causing it). Then look at that file and edit it to remove the error if you dare! (actually it is not that hard to see errors in instantiation files, because the IBM compiler just adds to them and never removes from them. That is, you may have redefinitions, which you can correct by removing the offending line.) OR, do a "qmake -f MyTestFile".

3.3 What to do if no error is reported, but Unresolved symbols are dumped?

Find the origin of unresolved symbols; if they are Aips++ symbols you must include the .h file relating to those symbols. If the symbols are non Aips++, it is a new library that has to be added to the list of libraries in your site makedefs.