The Instadia maker user's guide

Philip Bergen (pb@instadia.net)

May 14, 2001

Contents

1 Introduction **2**
- 1.1 About this document . . . 2
- 1.2 Versions . . . 2

2 Making the maker **4**
- 2.1 Files needed . . . 4
- 2.2 Make it . . . 4

3 Using maker **5**
- 3.1 Philosophy and requirements to the source . . . 5
  - 3.1.1 Limitations . . . 5
  - 3.1.2 Result . . . 6
- 3.2 Options . . . 6
  - 3.2.1 MAINFILE . . . 6
  - 3.2.2 MAKESCRIPT . . . 6
  - 3.2.3 EXECUTABLE . . . 6
  - 3.2.4 LINKER . . . 6
  - 3.2.5 LINKER_TAIL . . . 7
  - 3.2.6 COMPILER . . . 7
  - 3.2.7 DOTOUTPUT . . . 7
  - 3.2.8 NOFILE . . . 7
  - 3.2.9 NOLINK . . . 7
  - 3.2.10 G2EXCLUDE . . . 8
  - 3.2.11 MAKER_ADD_DEPENDENCY . . . 8
- 3.3 Macros . . . 8
  - 3.3.1 $DATE$ . . . 8
  - 3.3.2 $TIME$ . . . 8

4 Dot output **9**
- 4.1 What is dot? . . . 9
- 4.2 Getting dot . . . 9
- 4.3 Generating the graphs . . . 9
- 4.4 What the graphs show . . . 9

5 Appendix **10**
5.1 magic.sh 10

Chapter 1

Introduction

1.1 About this document

This document addresses any developer who is tired of writing makefiles. The maker allows compilation of a complex (but with some limits) C or C++ project.

Throughout this document files are either source files (*.cpp, *.cc or *.c) or header files (*.h, *.hh or *.hpp) or they are object files (*.o or *.obj).

The maker cannot substitute makefiles entirely but its a good attempt at it. You might end out with a makefile calling maker but then the makefile will be short and easy to write and you'll know that you'll always compile all files that have such dependencies and not any others.

The optimal compilation awaits!

1.2 Versions

- `$Revision: 1.3 $`
- `$Author: pbergen $`
- `$Date: 2001/05/13 22:01:16 $`

```
$Log: maker.tex,v $
Revision 1.3  2001/05/13 22:01:16  pbergen
Included option G2EXCLUDE that was added to provide means to
simplify the very useful .class graph.

Revision 1.2  2001/05/10 05:54:57  pbergen
Corrected a few typoes and missing files.

Revision 1.3  2001/05/09 14:53:38  pb
Added NOLINK option.
```

```
Revision 1.2  2001/05/09 12:11:06  pb
Added new directive MAKER_ADD_DEPENDENCY.

Revision 1.1  2001/05/08 16:44:52  pb
The documentation to the maker.
```

Chapter 2

Making the maker

2.1 Files needed

You need these files: `maker.cpp`, `maker.h`, `makeutilities.h`, `optionreader.h` and `file.h`.

This manual might come in handy too: `maker.tex`, `maker.ps` or `maker.pdf`.

2.2 Make it

Acknowledging the hatred of make files all you need as a decently recent make or a C++-compiler and either of these should get you going:

```
make maker
```

or

```
gcc maker.cpp -o maker
```

Chapter 3

Using maker

3.1 Philosophy and requirements to the source

Maker works this way: you supply the main source file as MAINFILE and all the header files you can find.

Then the maker looks in the MAINFILE for include statements and if the file being included was among the header files supplied on the command line. Files that match are added as dependencies to MAINFILE. These added files are evealuated in the same way as MAINFILE generating dependencies to them and so on. Anyone thinking recursion? -You bet!

Everytime a header file is added the maker tries to guess an appropriate source file matching this header (by changing the extension to .cpp, .cc and .c). If there is such a file it is added as dependency to the header and it too is scanned for include statements.

Looking for include statments stops after 256 lines or when the keyword `class` is encountered. No preprocessing is made all non include statements are simply ignored.

3.1.1 Limitations

The above leads us to some limits of the maker:

Include statments must look like this: `#include` and not `# include`. This is not a bug. If you have include-statements that you want the maker to ignore you can just add that space between `#` and `include` and the maker will ignore the include (but the compiler will not!).

If you have more than 256 lines of include statements in your source you will have to raise that level in `file.h` look for `static const int MAX=256;` and recompile the maker.

Each source file must have a header file and that header file must be included by a file within the dependencies of MAINFILE. If you have one header file that lists functions defined in more than one source file

then that will not compile. **Exception:** Dependencies can be added with MAKER_ADD_DEPENDENCY. See below.

3.1.2 Result

After running the maker the result is a shell script that tries to compile the files one by one. When a compilation fails the script halts. At the end of each script the final linking takes place.

3.2 Options

The maker takes the following options (must be CAPITALS) as command line parameters (preceded by a minus '-') or in an options file.

Siimple macros can be used as well. These are described below.

3.2.1 MAINFILE

This option specifies the main source file.
E.g.: MAINFILE=maker.cpp
Default: If there was an optionsfile the default is the options file without extension, else the maker falls out with an error.

3.2.2 MAKESCRIPT

This option specifies what to call the shell script.
E.g.: MAKESCRIPT=build.sh
Default: If there was a MAINFILE the default is the MAINFILE with extension `.sh`, else the maker falls out with an error.

3.2.3 EXECUTABLE

This option specifies what to call the resulting executable.
E.g.: EXECUTABLE=maker
Default: MAINFILE without extension.

3.2.4 LINKER

This option specifies what command to run as linker.
E.g.: LINKER=g++ -lm -lpng -lz -Wl,–rpath -Wl,/usr/local/lib
Default: g++ -lm

3.2.5 LINKER_TAIL

This option specifies what to append to the linking command after all the object files.
E.g.: LINKER_TAIL=myRPC/libmyRPC_s.a
Default: empty.

3.2.6 COMPILER

This option specifies what command to run as the compiling command.
E.g.: g++ -g -Wall
Default: g++ -O2 -g

3.2.7 DOTOUTPUT

This option specifies that you want dot output and what the files should be called. There will always be two files: `yourname.dot` and `yourname.all.dot`. See the dot chapter for more details.
E.g.: dot.dot
Default: nothing.

3.2.8 NOFILE

This option has two purposes. When used as a command line parameter it tells the maker that none of the parameters are the name of an option file. The nifty feature is when it is used within a source file. If a source file is equipped with a comment containing maker options as lines in an option file, no option file will be necessary to build you project even if it needs to set some of the parameters! When the maker sees the NOFILE option it stops parsing the source file (or option file). But since it was told that there was no option file it assumes that the file on its command line is actually a MAINFILE.
E.g.: look in `maker.cpp`...
Default: nothing.

3.2.9 NOLINK

This option tells maker not generate a linking command in the make-script.
E.g.: -NOLINK
Default: nothing.

3.2.10 G2EXCLUDE

This option tells maker to exclude a file from the .class dot-file. Its main purpose being to provide means to reduce cluttering of this very useful type of graph.

This option can be inserted many times to exclude more than one file.

The file name must be entered without path.

E.g.: -G2EXCLUDE=intlist.cpp -G2EXCLUDE=logger.cpp
Default: nothing.

3.2.11 MAKER_ADD_DEPENDENCY

This option is source/header file only. It will not work in an option file or as a command line parameter. It adds a source-file dependency to the file it is written in. This file needs not be included on the list of files on the command line. Useful when one header file encapsulates the functions of many source files. Make sure you hide the statement inside comments but with no white-space to the left of it.
E.g.: MAKER_ADD_DEPENDENCY=“somefile.cpp”
Default: nothing.

3.3 Macros

Macros can appear anywhere in any option and will always be expanded as follows below.

3.3.1 $DATE$

The DATE-macro is expanded to the current date in this format:
`yyyy-mm-dd`

3.3.2 $TIME$

The TIME-macro is expanded to the current time in this format:
`hh:mm`

Chapter 4

Dot output

4.1 What is dot?

Dot is a fabulous graph-drawing utility made by AT&T and Bell labs (thanks a million guys!). It draws graphs of related nodes based on a simple text file. You need not tell where to place the nodes, that's what dot takes care of.

4.2 Getting dot

Dot is part of the graphviz package available at this URL:
`http://www.research.att.com/sw/tools/graphviz/`.

4.3 Generating the graphs

If the dot output was named `dot.dot` then the following commands renders postscript files:

```
dot dot.dot -Tps -o dot.ps
dot dot.all.dot -Tps -o dot.all.ps
```

If the desired output is a PNG-file do the following:

```
dot dot.dot -Tpng -o dot.png
dot dot.all.dot -Tpng -o dot.all.png
```

4.4 What the graphs show

The one with `.all.` in it, contains all header and source files found and how they relate. The other is a map of how the source files only are related.

When a source file needs to be rebuilt it is denoted by a shaded background in the box.

Chapter 5

Appendix

5.1 magic.sh

The magic script is a simple way of building a file in your project if you have a standard option file:

```
maker standard.opt -MAINFILE=$* $(find .  -name "*.h")
sh build.sh
```

Now to compile a file in your project you only need to write:
`magic.sh somepart/part1executable.cpp`.