PANDORA PRODUCTS

Software Manual pbtomake

Author Jim Schimpf

Document Number: PAN-200603011
Revision Number: 0.6
24 Oct 2006

Pandora Products.
215 Uschak Road
Derry, PA 15627

Pandora Products.

215 Uschak Road

Derry, PA 15627

Phone: 724.539.1276

Web: http://members.bellatlantic.net/~vze35xda/

Email: vze35xda@verizon.net

Document Revision History

Use the following table to track a history of this documents revisions. An entry should be made into this table for each version of the document.

Version	Author	Description	Date
0.1	js	Initial Version	15-Mar-2003
0.2	js	Convert to LyX form	11-Mar-2006
0.3	js	Complete Code description	12-Mar-2006
0.4	js	Add Clean & Install (Florian Wagner)	8-May-2006
0.5	js	Add target and object directory	13-Oct-2006
0.6	js	Add more infor on target generation	25-Oct-2006

Contents

1 Preface

pbtomake is a command line utility designed to read the structure of a XCode project and build a corresponding makefile. This is useful when you are collaborating with Linux/UNIX users and want to share code . You can use XCode on the Mac then give them the code with the makefile and they can build it on their systems.

XCode evolved from an earlier program called Project Builder so this program's name started as Project Builder to Make (pbtomake) and never got changed in the transition to XCode.

The program has also been changed to produce makefiles for both executables and dylib type projects. This will be handled automatically and it will be the correct makefile for the type of target. Also it has been changed to handle multiple target make files (see 3.2.9 on page 5)

2 Operation

2.1 Project Preparation

Before starting the conversion you must go through your XCode project and make all file references **Project Relative**. Do this by selecting the files in the project and picking Show Info under Project. You will have a drop down menu to allow you to change the reference. This must be done as pbtomake depends on this style of reference to build the make file. (Note when you build projects if you start adding files this way it will be done for you by XCode)

Also ensure that any libraries added to your project are of file type archive or archive.ar. Do Get Info on the library and note both Path Type and File Type.

2.2 pbtomake installation

You will find the executable **pbtomake** in this directory. Install this so it is in your Terminal execution path. That is if your path includes ~/bin then put it there. In any case this file is all you need for the conversion.

2.3 Use

2.3.1 Running pbtomake and errors

Open a terminal window and cd to where your <my project>.xcodeproj lies. Then all you have to do is:

```
pbtomake -i <my project>.codeproj
```

This will run and if there are no errors will create a makefile right in this directory. The makefile will reference all it's files relative to this directory so is designed to be stored right here and be used here. If there are errors like:

```
** Fix this file and run again
       ** GROUP FILE REF [delcomdrv.c]
-- Parse Done --
** BAD PROJECT FILE FIX AND TRY AGAIN
```

it means that file (delcomdrv.c) has a group relative not a project relative reference in the project. Go into the project, Get Info on that file and change the reference to project relative. Running pbtomake again you should get

```
-- Parse Done --
```

if there are no more incorrect references then the makefile is created right here.

2.3.2 Testing

You can test the makefile at this point by typing **make** and it will build your project just like XCode. All the .o (object) files will be created in this directory and the executable will be created right here also.

```
make
/usr/bin/cc  main.c -c -I. -o main.o
/usr/bin/cc  delcomdrv.c -c -I. -o delcomdrv.o
:
/usr/bin/cc -framework IOKit -framework CoreFoundation \
       main.o\
       delcomdrv.o\
       -o MyUSB
```

If all of this works then the makefile is complete and ready for modification for other systems.

2.3.3 Install & Clean

Florian Wagner has added support in the output makefiles to do clean and make. If you type make clean then all the object files from the make will be deleted allowing you to do a full build. If you type make install then the product of the make (i.e. the executable) will be copied to a specified directory (see -bin_dir 3.2.5 on page 4).

3 Building makefiles for other systems

3.1 Introduction

The previous section showed how to create a makefile that works in OS X. This is a useful check to see if the makefile finds all the project files and links to the correct libraries, you should do this before you consider building a makefile for say Linux or some other UNIX.

After you are satisfied with the makefile then you have to change it. Linux and other UNIX systems usually have different names for the C compiler and almost always have different linking conventions for the system libraries. Remember frameworks (on OS X) don't exist elsewhere so they will NOT be in your final makefile for non-OSX systems. Also when you write your code you might use defines like LINUX_SYS to turn on or off sectons of code that only are used in say Linux.

pbtomake has command line options to allow you to input special compiler options and link options so you can build a special purpose makefile for these other systems. I have found that I will make a small script file with all the parameters for a particular system make file for pbtomake. Calling this say **buildmakelinux.sh** then if my project changes I can just run the script again to create the makefile for that system.

3.2 Command line syntax

If you type pbtomake -v it will show you the program version and the options availble:

```
pbtomake PB Project -> makefile converter Ver: Oct 13 2006 07:49:26
Syntax pbtomake -i <Project.pbproj> [-o <fname>] [-cc <c compiler>] [-v]
         -i <fname>      Input Project (pbproj) DEF NONE
         -o <fname>      Output File (makefile)   DEF makefile
         -cc <compiler>  Compiler used DEF /usr/bin/gcc
         -cc_opt <compiler optons>       Compiler options DEF NONE
         -bin_dir <install directory>    Target directory for
                                         "make install" DEF NONE
         -obj <Obj directory>            Object files here DEF=".”
         -link_opt <link options>        Link stage options DEF NONE
         -no_framework   Supress -framework lines DEF Show frameworks
         -t <target>     Specify target name in multi-target Project files
         -debug          Turn on Debug output
         -v              Show this help
```

3.2.1 -i <Project.xcodeproj>

This specifies the project you want to convert, this is a required input and there is no default.

3.2.2 -o <makefile name>

This is the name of the makefile you wish to produce, if this is not input **makefile** is used.

3.2.3 -cc <compiler>

This is the name of the c or c++ compiler used. By default it uses /usr/bin/cc. If you are building a c++ project you will have to set this to /usr/bin/g++ on OS X. This name will vary depending on the target system. In general Linux and OS X use the same names (i.e. gcc) but others like Solaris and HP-UX have other names for their C compilers.

3.2.4 -cc_opt <Compiler options>

These are options you wish to pass to the C compiler. These can be things like -DLINUX_SYS=1 where you wish to use these defines to enable/disable code for that particular system. If you have a number of these like -DLINUX_SYS=1 -DTEST=1 then you must enclose them in quotes on the command line like:

```
pbtomake -i tst.xcodeproj -cc_opt ‘‘-DLINUX_SYS=1 -DTEST=1’’
```

or only the first one will be used

3.2.5 -bin_dir <Installation Directory>

This will specify where the output file will be copied when you do make install. By default none is specified. Also you must have write priviledges to this directory or the make install will not work.

This was added by Florian Wagner.

3.2.6 -obj <Object Directory>

This sets the make file to build the .o files in a specified directory. Doing this will eliminate the clutter of having all the .o files mixing with the executiable and source files. The clean is also modified to delete them from this directory. NOTE: This directory must exist, the make file created does not create this directory when it runs.

3.2.7 -link_opt <Link options>

Thees are options for the link phase. For OS X you generally don't need any options here. In the case of other systems you might need things like -lmath or -lpthread depending on your program.

Note again like the -cc_opt lines if you have more than one of these enclose the whole set in double quotes to ensure it is all passed to pbtomake.

3.2.8 -no_framework

If present will suppress the -framework links in the constructed makefile link options. These are needed for OS X builds but are NOT present in other systems. For testing you will leave this off, building an OS X compatible makefile. For your foreign system makefile you will probably want to use this option.

3.2.9 -target <Target Name>

XCode project files can have multiple targets i.e. they can build both an dylib and executable in the same project. This flag allow you to specify which of the targets in the project will be built by the makefile. In multiple targeted projects when this flag is not used makefile will usually make the target that occurs first in the project.

3.2.10 -debug

Will output the internal tables of the program as it runs. Has no effect on the created makefile but is used for diagnostics in case of problems.

3.2.11 -v

Will cause pbtomake to output this syntax listing and stop. Will not create any output file.

3.3 And then what

You have to move your code to the other system and try this out. It might not work, at that point see if you can fix the problem with options in pbtomake and regenerate the makefile and try again. There will be times this is not possible. Then you will have to use pbtomake to build a makefile and edit it manually to fix it for the target system.

I have only tried pbtomake makefiles, moving code to Debian Linux, Solaris and HP-UX so I am sure there are many more variations. If you have a persistant problem with certain systems contact me with specifics and we can see if changes in pbtomake can solve them. Also source code is included so you can build your own special purpose version of pbtomake.

Once the makefile is created it usually isn't changed in major ways for the course of a project so even if you have to do some rather complicated editing, you only have to do that once. Use pbotmake to create an almost useful makefile, doing all the busy work for you then if necessary you can apply the final editing for a particular system.

4 Source Code

4.1 Introduction

The source code is included in the project and this section will describe the major modules and how it fits together. If you wish to modify the program the XCode project is here and you can easily create special purpose versions of the program to build YOUR makefile.

Originally this was designed to work with Project Builder files, the predecessor to XCode. These files were hugely more complicated than now. XCode 2.0 changed the project structure and simplified it greatly. The makefiles produced by the program for these older versions were not particularly good, they would work for a simple project but fail on larger ones. With XCode 2.0 and later, with the simplified internal structure, the output makefiles are much improved and will work for even quite complicated projects (my test is a $>$ 350 file simulator project and it now works.)

4.2 Project File Data

The project file has a section (in the internal file project.pbxproj in the project bundle)

```
/* Begin PBXFileReference section */
          3870AEE909744652008D4F3B /* ctcpnet.c */ = {isa = PBXFileReference; fileEncoding = 30;
lastKnownFileType = sourcecode.c.c; name = ctcpnet.c; path = ../POSIX/ctcpnet.c; sourceTree = SOURCE_ROOT; };
          3870AEEA09744652008D4F3B /* ctcpnet.h */ = {isa = PBXFileReference; fileEncoding = 30;
lastKnownFileType = sourcecode.c.h; name = ctcpnet.h; path = ../POSIX/ctcpnet.h; sourceTree = SOURCE_ROOT; };
```

with lines like the above. The program reads these lines and these contain all the information needed to build the make file.

As you can see each line consists of a number of key value pairs which describe the type of file (include, source, archive or framework) and its path. The program opens this file and reads the lines trying to find this section. Then each line in this secton is read in and analyzed.

4.3 Source Files

In the project the main.c gathers in the command line options, starts the analysis process and dumps the output makefile if input is correct. There are two other support files needed here, first CMaker.ch which does most of the operations second CpbxLexFile.cp which is used to parse the project data.

4.4 Multiple Targets

When building from mulitple targeted project files pbtomake takes a very simple approach. It builds a make file with ALL the include files mentioned in the project but only the code (c,cpp) files associated with the particular target. This way if you have include files that belong to both targets the compile will find them. A new module (target.c) was added to support this. It will search the project file to find PBXNativeTarget section that has a target matching that found on the command line. In this section is a list of the code files belonging to the project. This list is converted to a hash table and in the make file generation each code file is matched to the list and only those in the list have link lines and compile lines generated. This way the makefile only builds the code needed for the particular target.