Writing GNUstep Makefiles

Nicola Pero nicola.pero@meta-innovation.com

First Edition: June 2000; Last Updated: February 2010

1 What is it

The GNUstep make package is aimed at providing a simple and automatic way to manage compilation of GNUstep projects on different machines and environments. Your project will remain completely portable to any platform running GNUstep without the need to (explicitly) use complex packages such as autoconf or automake.

This document is for GNUstep make version 2.2.0 or above.

2 A First Tool

Let's try it out by making a little command line tool using the GNUstep make package. Let's start by creating a directory to hold our project. In this directory, type the following extremely simple program in a file called say `source.m`.

```
#import <Foundation/Foundation.h>

int
main (void)
{
  NSLog (@"Executing");
  return 0;
}
```

The function `NSLog` simply outputs the string to `stderr`, flushing the output before continuing. To compile this little program as a command line tool called `LogTest`, add in the same directory a file called `GNUmakefile`, with the following contents:

```
include $(GNUSTEP_MAKEFILES)/common.make

TOOL_NAME = LogTest
LogTest_OBJC_FILES = source.m

include $(GNUSTEP_MAKEFILES)/tool.make
```

And that's it. At this point, you have all the usual standard GNU make options: typically make, make clean, make install, make distclean. For example, typing `make` in the project directory should compile our little tool. It should create a

single executable `LogTest`, and put it in the subdirectory "obj" of your current directory, that is, in

```
./obj
```

You can try the tool out by typing

```
./obj/LogTest
```

When you type `make`, the system will check if any changes were made to the source code since the last time you compiled, and if so, it will rebuild your software.

If you want to remove the results of a previous compilation (for example, to later force a recompilation with different options) you can use the command

```
make clean
```

3 Installing the Tool

To install the tool, simply type `make install`; you usually need to be root to install the tool on a system directory. The tool will be installed by default in the "LOCAL" domain, which usually means `/usr/local/` in a Unix FHS filesystem layout, or `/usr/GNUstep/Local` in a traditional GNUstep filesystem layout. Which filesystem layout is used on your system depends on how GNUstep was configured on your system. You can change the installation domain as follows:

```
make install GNUSTEP_INSTALLATION_DOMAIN=USER
```

This will install the tool in your own user GNUstep directory (eg, `/home/nicola/GNUstep`), which doesn't require you to be root and could be a better place for testing. I often do this when testing my own code and programs, and it is very handy.

If you are packaging your tool and want it to be installed in the "SYSTEM" domain (ie, `/usr` in an FHS filesystem layout and `/usr/GNUstep/System` in a GNUstep one), you would do:

```
make install GNUSTEP_INSTALLATION_DOMAIN=SYSTEM
```

4 Enabling Verbose Messages

By default, GNUstep make only prints short messages about what it is doing. If you want to see the actual commands being executed, you can add `messages=yes` on the command-line, as in:

```
make messages=yes
```

You can use the option with any target, such as 'install' or 'clean'. For example, if you're not sure where your tool is being installed, you can use

```
make install messages=yes GNUSTEP_INSTALLATION_DOMAIN=USER
```

to find out.

5 Disabling Debugging

By default, executables are created with debugging enabled. This means that they are created with debugging symbols, i.e., compiled with the `-g` option (which is useful for debugging it with gdb), and compiled using the `-DDEBUG` compiler flag (which defines the preprocessor symbol `DEBUG` during the compilation). In this way, you may isolate code to be executed only when compiling with the debug option typically as follows:

```
#ifdef DEBUG
  /* Code compiled in only when debug=yes */
#endif
```

To compile this tool with debugging disabled, type:

```
make debug=no
```

Of course, if you have already compiled your tool with debugging enabled, you need to do a `make clean` first to remove the previous compilation, then type `make debug=no` to compile again with debugging disabled.

6 A first App

Let's try now to compile an application. Modify our source file `source.m` to read

```
#import <Foundation/Foundation.h>
#import <AppKit/AppKit.h>

int
main (void)
{
  NSAutoreleasePool *pool;

  pool = [NSAutoreleasePool new];

  [NSApplication sharedApplication];

  NSRunAlertPanel (@"Test", @"Hello from the GNUstep AppKit",
                   nil, nil, nil);

  return 0;
}
```

(Ignore the autorelease pool code for now - we'll cover autorelease pools in detail later). The line containing `sharedApplication` initializes the GNUstep GUI library; then, the following line runs an alert panel. To compile it, we rewrite the GNUmakefile as follows:

```
include $(GNUSTEP_MAKEFILES)/common.make

APP_NAME = PanelTest
```

```
PanelTest_OBJC_FILES = source.m

include $(GNUSTEP_MAKEFILES)/application.make
```

And that's it. To compile, type in `make`. The result is slightly different from a command line tool. When building an application, the application usually has a set of resources (images, text files, sound files, bundles, etc) which comes with the application. In the GNUstep framework, these resources are stored with the application executable in an 'application directory', named after the application, with `app` appended. In this case, after compilation the directory `PanelTest.app` should have been created. Our executable file is inside this directory; but the correct way to run the executable is through the `openapp` tool, in the following way:

```
openapp ./PanelTest.app
```

(openapp should be in your path; if it is not, you should check that GNUstep is properly installed on your system).

7 Debugging an Application

Debugging an application is quite simple. Applications, like tools and everything else, are compiled with debugging enabled by default; to debug the application, use

```
openapp --debug ./PanelTest.app
```

This will run `gdb` (the GNU debugger) on the executable setting everything ready for debugging.

8 Preamble and Postamble

You may happen to need to pass additional flags to the compiler (in order to link with additional libraries, for example) or to be willing to perform some additional actions after compilation or installation. The standard way of doing this is as follows: add a file called `GNUmakefile.preamble` to your project directory. An example of a `GNUmakefile.preamble` is the following:

```
ADDITIONAL_OBJCFLAGS += -Wextra
```

This simply adds the `-Wextra` flag when compiling, which causes GCC to print a lot more warnings than it would normally do. In general, you would use a `GNUmakefile.preamble` to add any additional flags you need (to tell the compiler/linker to search additional directories upon compiling/linking, to link with additional libraries, etc).

Now, you would want your `GNUmakefile` to include the contents of your `GNUmakefile.preamble` before any processing. This is usually done as follows:

```
include $(GNUSTEP_MAKEFILES)/common.make

APP_NAME = PanelTest
```

```
PanelTest_OBJC_FILES = source.m

include GNUmakefile.preamble
include $(GNUSTEP_MAKEFILES)/application.make
```

The most important thing to notice is that the `GNUmakefile.preamble` is included *before* `application.make`. That is why is called a preamble.

Sometimes you also see people using

```
-include GNUmakefile.preamble
```

(with a hyphen, `-`, prepended). The hyphen before `include` tells the make tool not to complain if the file `GNUmakefile.preamble` is not found. If you want to make sure that the `GNUmakefile.preamble` is included, you should better not use the hyphen.

If you want to perform any special operation after the GNUmakefile package has done its work, you usually put them in a `GNUmakefile.postamble` file. The `GNUmakefile.postamble` is included *after* `application.make`; that is why is called a postamble:

```
include $(GNUSTEP_MAKEFILES)/common.make

APP_NAME = PanelTest
PanelTest_OBJC_FILES = source.m

include GNUmakefile.preamble
include $(GNUSTEP_MAKEFILES)/application.make
include GNUmakefile.postamble
```

Here is a concrete example of a `GNUmakefile.postamble`:

```
after-install::
        $(MKINSTALLDIRS) /home/nicola/SpecialTools; \
        cd $(GNUSTEP_OBJ_DIR); \
        $(INSTALL) myTool /home/nicola/SpecialTools;
```

(make sure you start each indented line with `TAB`). This will install the tool `myTool` in the directory `/home/nicola/SpecialTools` after the standard installation has been performed.

You rarely need to use `GNUmakefile.postamble`s, and they were mentioned mainly to give you a complete picture.

9 Further Reading

For further examples and information on `GNUmakefile`s, you may want to have a look at the various test apps and tools in the GNUstep core library. There is also another more advanced Tutorial covering libraries and aggregate projects.