Building and Linking Against MOOS

Paul Newman

March 1, 2009

Abstract

This document will tell you how to build the MOOS project and how to link your own applications against the MOOS libraries.

1 Introduction

In this document, we'll cover how to compile and link your MOOS Applications. The simplest way to do this is by using CMake and adding your own source tree as a sibling of the "Core" directory under the MOOS – this is described in detail in Section **??**. However you may not want to do this, preferring to handcraft your own Make files for each platform your application will be running on – the information you will require to do this is described in Section **??**.

1.1 Downloading the Source Tree

The `MOOS` can be downloaded from `http://www.robots.ox.ac.uk/~pnewman/TheMOOS` and also via `svn` [1].

All of the latest MOOS code is cross-platform. Built into the source tree is a cross-platform build system that relies on `CMake` – a third-party executable available from *www.cmake.org* or the MOOS website.

2 The Source Tree Shape

Figure **??** shows the shape of the MOOS tree in release packages.[2] In this section we'll survey the contents of each directory. We shall not however describe in any detail what the contents of a directory does when built or how it may be used — such details can be found elsewhere.

[1] Directions on how to use svn to grab releases can be found on `http://www.robots.ox.ac.uk/~pnewman/TheMOOS`.

[2] You obviously don't have to stick with this but it has some positives.

Figure 1: The MOOS source tree (as downloaded)

2.1 Core

The "core" directory contains code that has nothing per-se to do with robots. It is basically only concerned with the communications API. See the documenta-

tion on Programming with MOOS and the MOOSComms architecture for more detail. The minimal conceivable build would be these three projects: the two base MOOS libraries and the MOOSDB server that binds processes together.

2.2 Essentials

This directory also contains domain-neutral applications (i.e.not necessarily only useful for robotics projects) but they are things that tend to be used all the time by folk who have adopted MOOS. It is recommended that you invest time in learning how the processes work – they'll save you a lot of time.

2.3 Tools

The tools subdirectory is again largely domain-independent, but contains, especially in the Graphical subdirectory, some decidedly useful applications.

2.3.1 Graphical

Projects in this directory have a dependency on the FLTK cross-platform toolkit. `uPlayback` is a lightweight tool which can playback into a MOOS community file created by `pLogger` .`uMS` is a fantastically useful tool for spying on a multitude of processes talking to each other using the MOOS Communications API.

2.3.2 Simulation

This directory contains simulations for ocean- and land-based vehicles.

2.4 Matlab

This directory builds a very useful project. It builds a mexglx or dll file (depending on your OS) which allows Matlab scripts to behave as if they were fully fledged MOOS-enabled processes. It's pretty useful if you want a quick visualisation of data or to prototype some code. To build this project you do need to have Matlab installed.

2.5 NavigationAndControl

The `NavigationAndControl` directory contains projects that are common in mobile robotics apps — `pNav` `pHelm` and `iRemote` . The former two of these have a dependency on the `Newmat` numerical library which can be found in the `Thirdparty` directory. There is a case to put `iRemote` in the instruments subfolder as it interfaces to a "human instrument" – but it seemed more robot-centric rather than sensor-based — *sub-judice*.

2.6 Instruments

The instruments subtree is pretty obvious – note that it includes `iMatlab` which is a peculiar instrument as it provides an interface to Matlab not a device or human.

2.7 Thirdparty

The Thirdparty directory contains code that for the most part is written by third-party authors. Newmat is a linear algebra package and FLTKVW is an extension to the FLTK functionality with a few MOOS-Graphics additions.

3 Cross-Platform Building using CMake

The use of `CMake` allows a code-author to describe in a high-level way what should be built in a project. The important point from this is that it allows (via "meta-make" files called `CMakeLists.txt` found in every directory) Make files to be written for Unix and Microsoft developer studio development alike. This is a massive win in terms of cross-platform development / project management. [3]

By toggling the relevant fields in the ccmake curses-GUI you can turn on/off Makefile generation for different parts of the project. Pressing "c" to configure then parses the newly included CMakeLists.txt files and may present some new options for you to select. A basic build should work out-of-the-box simply by accepting the default values. But to be sure we'll now walk through the process step by step.

1. Download and install `CMake` for your platform (from `www.cmake.orgor` the MOOS website).
2. "cd" into the MOOS root directory.
3. Type `ccmake ./` – you should see something like Figure ??.

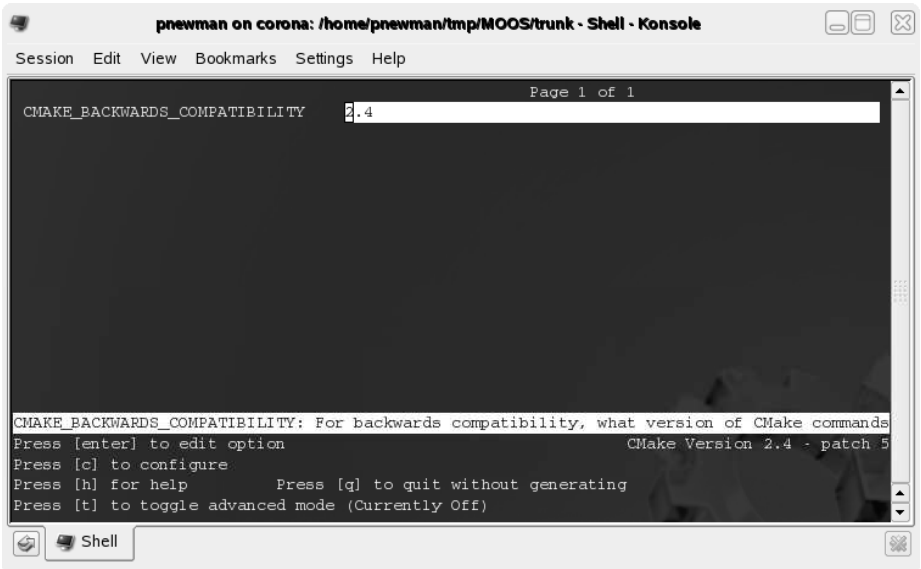

Figure 2: The initial CMake Screen - Win32 platforms have a gui-dialog rather than an ncurses interface

[3] Of course if one only develops for one OS it is hard to see the benefit.

4. Press “c” to configure – the screen should look like something like Figure **??**. *

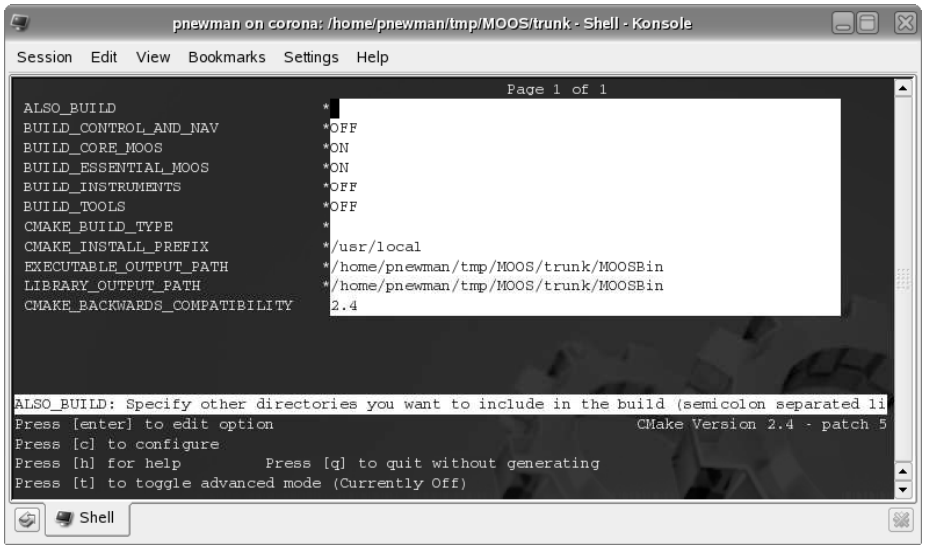

Figure 3: The initial CMake Screen after an initial configure – note no option to generate Make files is present as yet. Build directories and build options (i.e. what to build and what not) can be selected at this stage.

5. Press “c” to configure once more. The screen should now look something like Figure **??**.

6. Notice an option “g” has appeared saying we are good to go....Press “g” to generate the MOOS make files. If you have followed the above steps CMake will exit and you’ll see (along with one or two other CMake generated files) a master Makefile (or dsp/dsw/sln file for DevStudio builds) in the root directory.

7. You are now ready to build the project. If you have selected an IDE build, (like Devstudio or KDevelop) load the relevant newly created project file into your idea and perform whatever steps are needed to build. In Figure **??** I’ve just typed “make” in a linux terminal. I went with all the default options and now just “Core” and “Essential” projects are being built. I have a warm fuzzy about this.

Note that the Utils directory includes massively useful GUI applications which link against FLTK. You are expected to have the library and include files in your system path (this should have been done for you using the *configure, make, make install* paradigm when building the FLTK library)

3.1 Incorporating Private Subtrees

Entering a suitable directory or symbolic link in the `ALSO_BUILD` field causes CMake to recurse into that directory when creating Make files. This way developers can include their own projects in the build paradigm. For example,

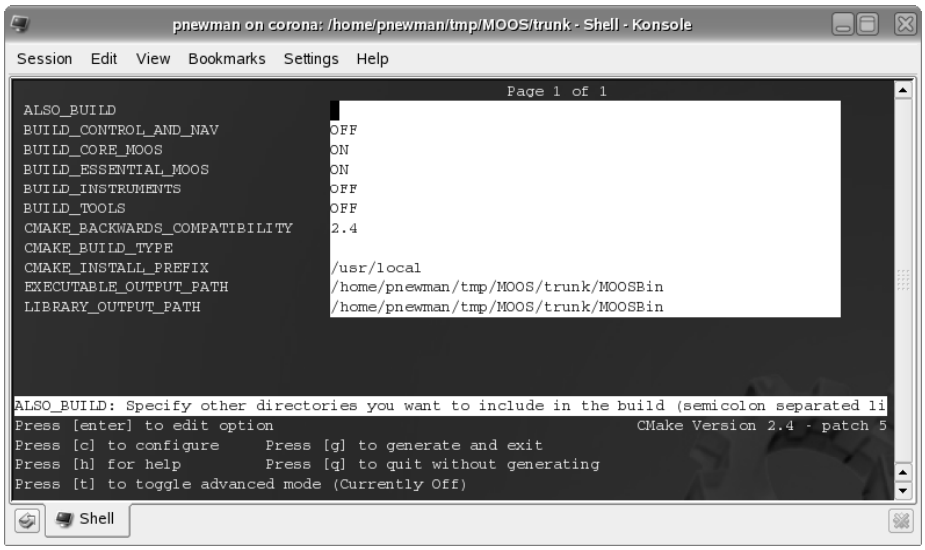

Figure 4: The final CMake Screen before generating the build files (".dsp" or Makefile — depending on platform). Please note "Configure" may have been selected several times before this final option is presented.

the Oxford Mobile Robotics group has a directory called OXMOOS which lives in a directory in MOOS (just like Core or Utils does). Typing OXMOOS in the `ALSO_BUILD` field causes CMake to recurse into the OXMOOS directory and follow the instructions in the CMakeLists.txt file therein (which causes it to recurse further down into each of our individual application projects). The advantage of this is that it inherits all the build flags and library settings of the parent MOOS project, and so things like the library paths (MOOSBin in the example above) are automatically passed to the compiler.

3.2 Where are the Binaries?

By default binaries and libraries are placed in *root*`/MOOSBin` . This of course can be changed simply by typing in your preferred destination during the build system set up using CMake.[4] See Figure **??**.

4 Living Without CMake

Some MOOS users may not wish to use the CMake infrastructure (a CMakeLists.txt file in each directory) or tree provided in the releases – preferring instead to use old trusted Make files they have written themselves for whatever platform they wish to develop/deploy on. This of course will work — the source in no way depends upon the build system. If this is your preferred method, then

[4] Devstudio users will find the binaries in debug/release/releasewithdebug directories within MOOSBin depending on the active project configuration selected in the IDE.

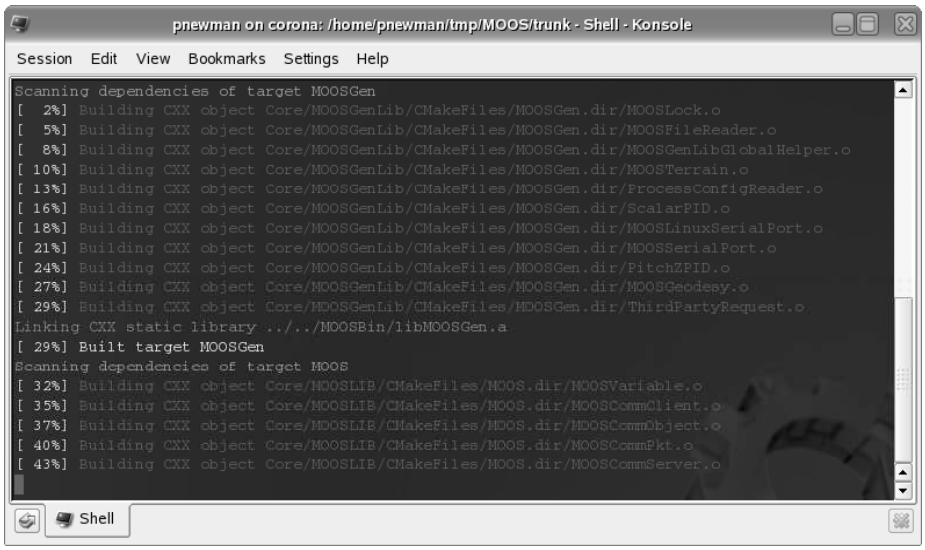

Figure 5: CMake has made the project files (which for me today in this example is a Linux "Makefile") and exited. Having typed "make" the build proceeds. The executables will be placed in the `MOOSBin` directory.

you will need to know which include paths are required and what libraries must be linked against.

4.1 Include Paths

To compile an application that uses the MOOS Comms API you need to add the following include paths.

root`/Core/`

Please note, header files are included relative to the *root*`/Core` directory for example by using `#include<MOOSLIB/MOOSCommsCLient.h>`

4.2 Library Paths

To link an application that uses the MOOS Comms API you need to add the following library paths

root`/MOOSBin` [5]

For historical reasons all binaries including libraries are placed in the `MOOSBin` directory [6]. You will also need to link against the libraries stated in Section **??**.

[5] Windows users – note Devstudio further qualifies this path with "Debug" or "Release" etc subdirectories

[6] This is something that needs looking at with separate `lib` and `bin` directories being preferable and planned.

4.3 Libraries to Link Against

To build an application that uses the core MOOS libraries, you will need to link against the following libraries (listed by OS).

Windows	**Linux**	**Solaris**
wsock32.lib	libm.a	libm.a
comctl32.lib	libpthread.a	libpthread.a
MOOS.lib	libMOOS.a	libsocket.a
MOOSGen.lib	libMOOSGen.a	libnsl.a
(console libs)		libMOOS.a
		libMOOSGen.a

If you want to roll your own Make files for the graphical applications inside the MOOS tree, look at the `CMakeLists.txt` files in the top root directory and the individual application directories to see what additional libraries are required.