SEPlib CMake update

Stewart A. Levin

ABSTRACT

SEPlib is now built with *CMake*. This report covers the futher changes made to the source tree in order to fully support Linux and Mac OS X platforms.

INTRODUCTION

As discussed in Levin and Clapp (2017), our SEPlib source code was converted this year to *CMake* for more rapid and less arcane build development and maintenance. The timing of the switch proved rather awkward for me as I had only recently figured out how to build shared libraries needed for use with Java under the prior system. Diving back into *CMake* arcana, I retackled the SEPlib build process as well as fixed several open bugs. In the following I share highlights, tricks, and tips of that adventure.

CMAKE LOOSE ENDS

The unfinished portions of the SEPlib cmake-based build were

1. shared libraries,
2. symbolic links, and
3. executable scripts.

Shared libraries

Shared libraries, also known as dynamic libraries, are the basis for modern computing paradigms such as browser add-ins. They are loaded into memory at program run time rather than being statically linked into the program executable. This puts some extra burden on the developer. First, unlike traditional object file libraries, accessing one symbol, for example a subroutine, in the shared library brings all the library's symbols into the program. Usually these include undefined symbols that must be pulled in from another library. This means that

- all dependent libraries need to be known when a shared library is built,

- the same symbol should not be defined in both a shared library and any dependent shared library, and
- there should be no circular references among the library and its dependent shared libraries.

Fortunately, the shared library work I had done under the older GNU build system provided the lists for the first item and already fixed the latter two items.

As it happens SEPlib shared libraries are not needed for any SEPlib programs and, indeed, are a nuisance to deploy and use. The only part of SEPlib currently relying on shared libraries is the subset supporting Java. For this reason, SEPlib is actually built twice, once using shared libraries and a second time using static libraries. This results in static executables and library pairs, one dynamic and one static.

Shared libraries are not completely eliminated from SEPlib with static executable linkage. Quite a few system and other external libraries may only be available in shared library format and, furthermore, may be incompatible between different machines and operating system versions. For this reason, I also scan the static executables and make copies of their remaining dependent shared libraries underneath the installed library directory to be use as last resort fallback options.

Symbolic links

Symbolic links are suprisingly awkward to create in *CMake*. They are only provided for Unix-based operating environments, e.g. Linux and Mac OS X, and implemented by invoking cmake separately with the command line option `-E create_symlink` and arguments giving the old and new names. Somewhat confusingly, this is not the same as `ln -s` but mimics instead `ln -s -r` so that to create

```
/opt/SEP/lib/NewName.a ---> /opt/SEP/lib/OldName.a
```

one should invoke the command

```
cmake -E create_symlink /opt/SEP/lib/OldName.a NewName.a
```

either separately or, as done in SEPlib, as a subprocess during the installation.

Executable scripts

Scripts play a different role in *CMake* than executables. For one thing, they may require configuration of some variables or strings. In addition, it is a mistake to install them with the

```
install(FILES scriptname DESTINATION bin)
```

cmake command as that relies on the script having the correct execute and read/write permissions in the source tree. One should use

```
install(PROGRAMS scriptname DESTINATION bin)
```

instead.

MISCELLANEOUS UPDATES

In addition to the three main areas discussed above, I also added support for FFTW high performance FFTs, resurrected older Motif-based 3D visualization programs, resolved a handful of vplot issues, added or modified external format converter utilities, and updated a number of program self-docs.

SUMMARY

While there are a few issues that still remain outstanding with respect to SEPlib builds and distribution, the current sources build quite cleanly on both Linux and Mac OS X and work under both GNU and Intel compilers.

REFERENCES

Levin, S. A. and R. G. Clapp, 2017, make, schmake: CMake: SEP-Report, **168**, 309–312.