Instructions for Including MESA Modules in MAESTRO

Ryan Orvedahl[1]
[1]Department of Physics and Astronomy, Stony Brook University

August 2, 2012

1 Introduction

Modules for Experiments in Stellar Astrophysics (MESA) is a collection of open source, robust and efficient modules developed for a wide range of applications in computational stellar astrophysics. The modules include macroscopic and microscopic physics such as nuclear reaction rates, opacity, equation of state, element diffusion data and atmosphere boundary conditions in addition to numerical routines. This document describes how to link the MESA modules to an outside program such as MAESTRO.

2 Setting Up MESA

To install MESA, download the source code from the MESA website, http://mesa.sourceforge.net. Check the website for the latest version, which is 4219 with the writing of this document. The codes and algorithms discussed here were created when the current version of MESA was version 4088. Between versions 4088 and 4219, there are minor changes that will cause some of the included routines to crash. To download the source code associated with version 4088, use the following command:

```
svn co -r 4088 http://mesa.svn.sourceforge.net/svnroot/mesa/trunk mesa
```

This will download the source into a directory called mesa located in the directory where this command was run. This step might take some time (~10 min depending on the internet connection) possibly because the final mesa directory will take up roughly 2.2 GB.

Rich Townsend has developed a Software Development Kit (SDK) to help install MESA called MESASDK. The SDK will install certain software packages that MESA uses and it will also change the PATH environment variable such that these programs become the default. A tarball of the SDK software and indepth instructions for its use are available on Townsend's website:

```
http://www.astro.wisc.edu/~townsend/static.php?ref=mesasdk
```

The instructions in this document do not use the SDK in order to avoid automatically changing the PATH variable.

2.1 Setup MESA Without MESASDK

2.1.1 Environment Variables (NOT using SDK)

A few environment variables need to be set (commands are shown for the Bourne shell):

1. Set location of mesa: `export MESA_DIR=<absolute-path-to-mesa-directory>`
2. Set OpenMP number of threads: `export OMP_NUM_THREADS=n`, a suggested value is the number of cores in the computer

2.1.2 Modify the Makefile (NOT using SDK)

The main makefile that contains compiler information including flags is located in the `mesa/utils` directory and is called `makefile_header_non_mesasdk`. Open it and change the following:

1. Compilers: First change the compilers labeled `CC` (C compiler) and `FC` (Fortran compiler) to the compilers of your choice (`MESA` defaults are `gcc` and `ifort`).
2. `UTILS_ISNAN`: Leave this as `utils_isnan_okay`, if `MESA` complains upon compiling the `mesa/utils` module, then change it to `utils_isnan_nope`.
3. `BLAS` and `LAPACK`: Set the location of `LAPACK` and `BLAS` provided with the `MESA` source tree using the following definitions:

```
WHICH_LAPACK = USE_SRCS
LOAD_LAPACK = -lmesalapack
WHICH_BLAS = USE_SRCS
LOAD_BLAS = -lmesablas
MKL_INCLUDE =
```

4. `PGPLOT`: Set the variable `USE_PGPLOT` to `NO` and set the variable `LOAD_PGPLOT` to be empty on the line corresponding to the compiler that was chosen. The default for `gcc` and `ifort` is `YES`, but for any other compiler the default is `NO`. `PGPLOT` controls the pop up images that are used in `mesa/star` (1D stellar evolution code).
5. `USE_SE`: Set the variable `USE_SE` to `NO` and set both `LOAD_SE` and `INCLUDE_SE` to be empty. `SE` is a library of formats used for writing/reading data in stellar evolution codes.
6. Compiler Flags: Installing `MESA` without OpenMP will result in a failed install. `MESA` is capable of being compiled in serial, but it must happen after the initial install. If not using `gcc` or `ifort` it will be necessary to enter all of the relevant compiler flags.

All `MESA` modules have a makefile that includes the main makefile named `makefile_header`. The file that was just edited was named `makefile_header_non_mesasdk` and so the name must be changed in order for the edits to take effect. This can be done as follows:

```
cp makefile_header_non_mesasdk makefile_header
```

At this point, `MESA` is ready to be installed, see Section 2.2.

2.2 Install `MESA`

If the environment variables were added to the `.bashrc` (or equivalent) file, do not forget to source that file before continuing. Now `MESA` is ready to be installed. Change directories to the `mesa` directory. Run the `install` script by typing `./install`. The permissions of the `install` script may need to be changed in order for it to be executable. This will untar many data tables used by the equation of state, network and chemistry modules. It will also compile and export all modules.

Following the execution of the `./install` script, `MESA` has been installed and the following has completed:

- SDK only: Supporting software (compilers, linear algebra packages, etc.) has been unpacked and installed
- Data tables have been untarred
- Linear Algebra packages have been compiled and linked
- All modules have been compiled (in parallel)
- All modules have been exported to the `mesa/lib` and `mesa/include` directories

If you want to run `MESA` in parallel, nothing further needs to be done. If you want to make sure all modules are compiled in serial, see Section 3, which addresses how to compile the modules.

3 Compile Modules Without OpenMP

In the mesa/utils directory, open the makefile_header file. Change the FCopenmp flag (gfortran default = -fopenmp) to be empty. This will disable OpenMP for all modules.

Anytime a change is made to a makefile or any source code, the module must be recompiled, which is a three step process involving the clean, mk and export scripts provided in each module directory. The first step is to clean out any old files using the ./clean script. Step two is to make the module using the ./mk script. The last step is to export the .mod files and the library to the right directories using the ./export script. Appendix A contains a simple script that is meant to help with compiling modules. It cleans, makes and exports every module and should be run in the top mesa/ directory.

3.1 Optional: Turn Off mesa/star OpenMP

There are four other places to turn off OpenMP, but they are all contained in the mesa/star directory, which is the 1-D stellar evolution code. If only using a few MESA modules that do not include the mesa/star stellar evolution code, then these last few places do not need to be modified in any way.

In mesa/star/private/micro.f there are three places to set the use_omp flag to False. They occur in the do_eos, do_neu and do_net routines. The fourth place to set the use_omp flag to False is contained in the do_kap routine in mesa/star/private/opacities.f.

4 Compiling MAESTRO to Use MESA

Compiling code that depends on MESA modules involves linking the code with the .mod files and libraries that are generated when MESA is compiled. After the export script has been run in all compiled directories, the .mod files and other supporting routines are copied to the mesa/include directory while the libraries are copied to the mesa/lib directory.

The MESA libraries have the name lib(module-name).a, e.g. libeos.a or libnet.a. The libraries need to be linked using the -L flag with the mesa/lib directory and a list of which libraries to load, e.g. -L$(MESA_DIR)/lib -leos -lnet -lnum. MESA .mod files are linked to code using the -I flag and the mesa/include directory, e.g. -I$(MESA_DIR)/include.

The order in which the libraries appear is important. If compilation errors related to undefined references to a particular MESA module such as eos_lib occur, check to make sure that the libraries are in the correct order. Of course it is also possible that an undefined reference really does exist.

5 Using MESA Modules

This section describes what needs to take place in order to use MESA modules such as setting up the module, invoking certain routines and shutting down the module. This discussion will only include information pertaining to the network and equation of state modules.

5.1 Equation of State

5.1.1 Set Up

There are several steps to setting up the equation of state:

1. Constant Initialization: This is done by calling the const_init subroutine
2. Data Directory: Define where the mesa/data directory is: This can be done by using the MESA_DIR environment variable that was set at installation
3. Chem Initialization: This is done by calling the chem_init subroutine: call chem_init(data_dir, 'isotopes.data_real', ierr). The second argument can also be 'isotopes.data_approx'. The "real" file contains actual values for atomic quantities such as atomic weight. The "approx" file contains approximate values for these quantities, e.g. the mass of the proton and neutron are both equal to 1.0.

The `ierr` argument is an integer that holds the exit status of the subroutine, non-zero means there was an error

4. EOS Initialization: This is done by calling the `eos_init` subroutine: `call eos_init(data_dir, eos_file_prefix, use_cache, ierr)`. `eos_file_prefix` is a character variable and should be declared and set equal to 'mesa'. `use_cache` is a logical variable that also needs to be declared and set equal to 'true'

5. Handle: Obtain a handle associated with the equation of state by calling the `alloc_eos_handle` function: `handle = alloc_eos_handle(ierr)`. `handle` should be declared as an integer

5.1.2 Calling the EOS

The call to the generic `MESA` equation of state looks like:

```
call eosDT_get(handle_eos, Z, X, abar, zbar, nspec, chem_id, net_iso, Xa, &
               rho, log10(rho), temp, log10(temp), res, d_dlnd, d_dlnT, ierr)
```

Where `Z` is the metallicity, `X` is the mass fraction of hydrogen, `abar` is the average atomic weight, `zbar` is the average atomic number, `nspec` is the number of species in the network, `Xa` is the array of mass fractions. See Section 5.2 for a complete description of `chem_id` and `net_iso`. The results of the EOS call are stored in the arrays `res`, `d_dlnd` and `d_dlnT`, which are all of size `num_eos_basic_results`. The `d_dlnd` and `d_dlnT` arrays are the derivatives of the `res` array with respect to density and temperature, respectively. The values held in the `res` array are shown in Table 1.

Output	Definition	Units
P_{gas}	gas pressure	ergs cm^{-3}
E	internal energy	ergs g^{-}1
S	entropy per gram	ergs g^{-1} K^{-1}
$dE/d\rho\|_T$		ergs cm^3 g^{-2}
C_V	specific heat at constant V $\equiv 1/\rho$	ergs g^{-1} K^{-1}
$dS/d\rho\|_T$		ergs cm^3 g^{-2} K^{-1}
$dS/dT\|_\rho$		ergs g^{-1} K^{-2}
χ_ρ	$\equiv d\ln P/d\ln\rho\|_T$	none
χ_T	$\equiv d\ln P/d\ln T\|_\rho$	none
C_P	specific heat at constant pressure	ergs g^{-1} K^{-1}
∇_{ad}	adiabatic T gradient with pressure	none
Γ_1	$\equiv d\ln P/d\ln\rho\|_S$	none
Γ_3	$\equiv d\ln T/d\ln\rho\|_S + 1$	none
η	ratio of electron chemical potential to k_BT	none
μ	mean molecular weight per gas particle	none
$1/\mu_e$	mean number of free electrons per nucleon	none

Table 1: `eos` output quantities and units

It is possible to call the HELM equation of state directly. This is done as follows:

```
call eos_get_helm_results(handle_eos, Z, X, abar, zbar, dens, log10(dens), &
                          temp, log10(temp), helm_res, ierr)
```

The `helm_res` array is of size `num_helm_results` and holds all of the output values from the call, including all derivatives. There are slight differences between the generic EOS call and the HELM EOS call: HELM does not need the mass fractions or the maps between `MESA` isotope indices and the mass fraction array indices (`chem_id` and `net_iso`, see Section 5.2). The HELM EOS returns on the order of 400 different quantities which include the values shown in Table 1 as well as quantities such as $dP/d(abar)$ and $dE/d(zbar)$.

5.1.3 Shutdown the EOS

There are two steps to shutting down the equation of state:

1. Free the handle: Call the `free_eos_handle` subroutine: `call free_eos_handle(handle_eos)`.
2. Shutdown: Call the `eos_shutdown` subroutine: `call eos_shutdown`

5.2 Mapping MAESTRO to MESA

MAESTRO and MESA refer to isotopes in a very similar way; using short names of the isotopes such as he4 and ni56. MAESTRO tends to use capital letters (He4 and Ni56) while MESA uses lowercase letters (he4 and ni56). A simple routine to convert from uppercase to lowercase will be needed.

In order to use MESA modules and obtain the correct result, it is important to make sure that MESA and MAESTRO agree on what isotope each entry in the mass fraction array refers to. This is done by using the `chem_id` and `net_iso` variables. Both variables are integer pointer arrays of size `num_chem_isos`, which is defined in one of the chemistry modules. `chem_id` maps from an isotope in the current net to the MESA data table that holds the properties of that isotope. `net_iso` is used to map from the MESA data tables to an isotope in the current network.

For example, assume there are 3 isotopes in the current net: H1, Be7 and C12. The `chem_id` array should look like this: `chem_id(:) = -1, chem_id(1) = 2, chem_id(2) = 18, chem_id(3) = 38`, and the `net_iso` array should look like this: `net_iso(:) = -1, net_iso(2) = 1, net_iso(18) = 2, net_iso(38) = 3`. Where 2, 18 and 38 refer to the MESA data table location of H1, Be7 and C12, respectively.

Care should be taken when setting up these arrays in order to avoid disagreement between the MESA mass fraction array and the MAESTRO mass fraction array.

5.3 Network

5.1.1 Set Up

There are significantly more steps involved when initializing the MESA network. Here is a brief outline of what needs to happen:

1. Get the MESA and AstroDev directory locations
2. Convert the short species names to MESA format
3. Generate a file that will tell MESA which reactions to add based on the isotopes in the current network (`call which_reactions`)
4. Initialize MESA `chem`, `weak` and `reaclib` modules
5. Setup the map between MAESTRO and MESA, see Section 5.2 for a more in depth explanation
6. Initialize MESA `rates` and `net` modules
7. Allocate a network handle
8. Call the `net_start_def`, `read_net_file`, `net_ptr` and `net_finish_def`
9. Output a log file containing the species in the current net and which reactions were chosen by MESA based on those species
10. Set `which_rates_choice` and allocate `which_rates` array
11. Set entire array equal to `which_rates_choice` and call `net_set_which_rates`
12. Initialize which linear algebra solver to use

After calling `net_set_which_rates`, the setup routine should call `net_setup_tables`. This is based on the example `one_zone_burn.f` program distributed with the `MESA` source code. I have not had a successful burn if the `net_setup_tables` routine is called at setup in the `setup_mesa_net.f90` program. I do have successful burns when it is placed within the burner program `Do_One_Zone_Burn.f90`. To prevent the setup routine from being called everytime a burn is to take place, I added a logical flag that signals if the `net_setup_tables` routine has alredy been called.

5.3.2 Calling the Network

Calling the `MESA` network from within the `MAESTRO` wrapper starts with calling a modified `MESA` wrapper:

```
results => Xout

call Do_One_Zone_Burn(dens, temp, dt, Xin, burn_ergs, Xout, results)
```

The inputs are density, temperature, how long to burn and starting mass fractions. The outputs are the energy generation rate (`burn_ergs`) in units of ergs g^{-1} s^{-1}, the final mass fractions and a pointer to the final mass fractions (`results`). The pointer is necessary because the `MESA` network works with pointers while `MAESTRO` uses standard arrays.

To calculate `rho_omegadot` ($\rho\dot{\omega}$) and `rho_Hnuc` ($\rho H_{\rm nuc}$) the following code can be used after a call to `Do_One_Zone_Burn`:

```
do i=1, nspec
   dX = Xout(i) - Xin(i)
   rho_omegadot(i) = dens * dX / dt
enddo

rho_Hnuc = dens * burn_ergs
```

There is a support routine called `burn_solout` that is passed to the `MESA` burner. This routine is called after every step of the burn and is responsible for calculating the energy generation rate. `burn_solout` is passed as an argument to the `net_1_zone_burn` subroutine (the `MESA` burner) and as a result the actual call to the `burn_solout` routine is buried deep within the `MESA` source code.

5.3.3 Shutdown the Network

Shutting down the `MESA` network requires only one step:

1. Deallocate `chem_id`, `net_iso` and `which_rates`

6 Install `MESA` on Hopper

Once `MESA` was installed and working properly on a local machine (e.g. Bender), I tried installing it on Hopper, which is a Cray XE6 machine. On Bender, which is an Intel(R) Xeon(R) X5650 machine, the compiler was `gcc` version 4.7.0, on Hopper I tried compiling `MESA` with the Cray Compiler using version 8.0.5. The module that was loaded to enable this was `PrgEnv-cray`.

There are several steps needed to compile `MAESTRO` with `MESA` on Hopper:

1. Edit the `makefile_header` in the `utils` directory
2. Edit the `build_and_test` script in the `utils` directory
3. Edit the `mtx` makefile
4. Edit the makefiles in `screen`, `utils`, `num` and `interp_1d`
5. Edit the `install` script in the main `MESA` directory

6. Edit the `BoxLib/Tools/F_mk/GMakedefs.mak` makefile

7. Edit the `BoxLib/Tools/F_mk/comps/Linux_cray.mak` file

8. Make sure the makefile `COMP` variable is set to `Cray` in the problem directory

6.1 `makefile_header`

In following the setup outlined in this document, I changed the compilers, used the source version of `LAPACK` and `BLAS`, turned off `PGPLOT` and turned off the `SE` formatting library. I also added the appropriate compiler flags. The results of what I changed are shown below:

```
# Cray C-compiler
CC = cc

# Cray Fortran-Compiler
FC = ftn

# if you need special flags for the compiler, define them here:
SPECIAL_FC_FLAGS = -h mpi0 -target=linux -emf
# -h mpi0 disables mpi optimization
# -target=linux specifies the build machine as linux based
# -emf: the m flag says create NAME.mod files, the f flag says
#         convert those to name.mod files

# must explicity define all compilation flags
FCbasic = $(SPECIAL_FC_FLAGS)
FCimpno = -eI
FCchecks =
FCwarn = -m 3
FCfixed = -f fixed -N 132
FCfixed72 = -f fixed -N 72
FCfree = -f free
FCopt = -O 1
FCdebug = -g -O0
FCstatic =

FC_fixed_preprocess = -eZ
FC_free_preprocess = -eZ

# Cray compilers have omp ON by default
FCopenmp =
# to disable omp:
#FCopenmp = -h noomp
```

6.2 `build_and_test` script

The `build_and_test` script in the `utils` directory is responsible for compiling the main module directory as well as compiling and running the test directory. We want to turn off the compiling and running of the test directory to ensure that `MESA` compiles successfully. A portion of the `build_and_test` script looks like this:

```
cd make
make
check_okay
cd ../test
./mk
check_okay
```

```
if [ -x test_quietly ]
then
        ./test_quietly
        check_okay
fi
./ck >& diff.txt
```

we wish to change it to this:

```
cd make
make
check_okay
#cd ../test
#./mk
check_okay
if [ -x test_quietly ]
then
#         ./test_quietly
        check_okay
fi
#./ck >& diff.txt
```

This disables the compiling and running of the test directory.

6.3 `mtx` makefile

In the `mtx/make/makefile`, there are a few compiler flags that were hard coded in. There is a `-w` flag in five places and a `-fno-common` flag in one place. The `-w` flag instances look like this:

```
%.o: $(MODULE_DIR)/lapack_src/%.f
      $(COMPILE_XTRA) -w $<

# must turn off optimization for dlamch or can get infinite loop!!!
dlamch.o: $(MODULE_DIR)/blas_src/dlamch.f
      $(COMPILE_XTRA_NO_OPT) -w $<
```

The `-w` should be deleted. There are three more instances immediately after these lines. The `-fno-common` flag appears here:

```
KLU_C = $(CC) -O3 -fno-common -fexceptions
KLU_I = -I$(KLU_DIR)
```

All three of the manually added flags (`-O3`, `-fno-common` and `-fexceptions`) should be deleted from the `KLU_C` line.

6.4 makefiles in `screen`, `num`, `utils` and `interp_1d`

When creating the library for these four modules, there is a `LIB_DEFS` variable in the makefile that holds the `*_def.o` file, e.g. `utils_def.o` or `num_def.o`. This files is never loaded into the library as seen below for the `utils` directory:

```
LIB = libutils.a
LIB_DEFS = utils_def.o
LIB_OBJS = $(UTILS_ISNAN).o utils_nan.o utils_dict.o utils_lib.o

$(LIB) : $(LIB_DEFS) $(LIB_OBJS)
      $(LIB_TOOL) $(LIB) $(LIB_OBJS)
```

To include the LIB_DEFS file:

```
LIB = libutils.a
LIB_DEFS = utils_def.o
LIB_OBJS = $(UTILS_ISNAN).o utils_nan.o utils_dict.o utils_lib.o

$(LIB) : $(LIB_DEFS) $(LIB_OBJS)
      $(LIB_TOOL) $(LIB) $(LIB_DEFS) $(LIB_OBJS)
```

This must be done for each makefile in the following directories: screen/make, num/make, utils/make and interp_1d/make.

6.5 install script

Certain modules that are unneccessary for use with MAESTRO do not need to be installed (and also cause some sort of installation error). In the main MESA directory, edit the install script and comment out the following four lines: do_one sample, do_one star, do_one adipls and do_one astero.

6.6 BoxLib Linux_cray.mak file

Two flags used in the compilation of MESA with the Cray compiler are important; the -e m or -em flag and the -e f or -ef flag. The -em flag tells the compiler to create .mod files of the form NAME.mod. The -e f flag, which must be used with the -em flag, instead outputs .mod files of the form name.mod. This is important for MESA because the export scripts copy the various libraries and .mod files to the correct directories. The export script in the const directory for example looks like this:

```
cp make/const_lib.mod ../include
cp make/const_def.mod ../include

cp make/libconst.a ../lib
cd ../lib
ranlib libconst.a
```

If the -ef flag is not enabled, the .mod file will be CONST_LIB.mod and will not get copied to the include directory (the library is unaffected). The default for MAESTRO is to not use the -ef flag, but it does use the -em flag. To make MAESTRO and MESA compatible, I added the -ef flag to the Linux_cray.mak file located in BoxLib/Tools/F_mk/comps. I also added a few more debugging flags:

```
FFLAGS   += -J $(mdir) -I $(mdir) -emf
F90FLAGS += -J $(mdir) -I $(mdir) -emf

ifdef NDEBUG
  FFLAGS   += -O 1
  F90FLAGS += -O 1
  CFLAGS   += -O 1
else
  FFLAGS   += -g -O0 -R bps
  F90FLAGS += -g -O0 -R bps
  CFLAGS   += -g -O0 -h bounds
endif
```

6.7 BoxLib GMakedefs.mak makefile

In MAESTRO, .mod files are located in the $(tdir)/m directory, where tdir = t/$(suf) and suf was in this case Linux.Cray. This means the MAESTRO routines are expecting to find module files with the name t/Linux.Cray/m/network.mod. The addition of the -ef flag affects the entire name of the module so the

Cray compiler was creating files with the name `t/linux.cray/m/network.mod`. To solve this, I modified what `suf` is defined to be when using the Cray compiler. This was done in the `BoxLib/Tools/F_mk/GMakedefs.mak` makefile by using a simple if statement:

```
ifeq ($(COMP), Cray)
  tdir = t/$(shell echo $(suf) | tr A-Z a-z)
else
  tdir = t/$(suf)
endif
odir = $(tdir)/o
mdir = $(tdir)/m
```

The shell command `echo $(suf) | tr A-Z a-z` has the effect of converting `$(suf)` to lowercase.

Appendices

A Compilation Scripts

Useful script written in `BASH` to help with compiling `MESA` modules. The script cleans the module directory, compiles the source code and exports the libraries and `.mod` files to the appropriate directories for every module.

A.1 Compile All Modules

```
#!/bin/bash
# Compile ALL mesa (v 4088) directories

function check_okay {
   if [ $? -ne 0 ]
   then
      exit 1
   fi
}
function do_one {
   for dir in "$@"
   do
     cd $1
     check_okay
     echo
     echo "Begin: " $1
     echo
     ./clean
     check_okay
     ./mk
     check_okay
     ./export
     check_okay
     echo "%%%%%%%%%%%%%%%%%%%%%%%%"
     echo " SUCCESS: " $1
     echo "%%%%%%%%%%%%%%%%%%%%%%%%"
     cd ..
   done
}
do_one adipls alert atm chem colors const diffusion eos
do_one interp_1d interp_2d ionization kap mlt mtx net neu
```

```
do_ont num rates reaclib screen star
if [ ! -e utils/mk ]; then
   cp star/mk utils/
fi
do_one utils weaklib
echo "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
echo "   Successfully Compiled All Modules"
echo "%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%"
```