UCSB, MESA SCHOOL 2012

MESA Refresher

The MESA council

August 12, 2012

1 Starting models and basic input/output

`MESA star` receives basic input from two Fortran namelist files. One file specifies the type of evolutionary calculation to be performed, the type of input model to use, the source of EOS and opacity data, the chemical composition and nuclear network, and other properties of the input model. The second file specifies the controls and options to be applied during the evolution.

There are two ways to start a new evolutionary sequence with `MESA star`. The first is to use a saved model from a previous run. A variety of saved models are distributed with `MESA` as a convenience. These saved models fall into three general categories: (1) Zero Age Main Sequence (ZAMS) models for $Z = 0.02$ with 32 masses between 0.08 and $100M_\odot$ (`MESA star` will automatically interpolate any mass within this range); (2) very low mass, pre-main sequence models for $Z = 0.02$ and masses from 0.001 to $0.025M_\odot$; and (3) white dwarf models for $Z = 0.02$ with He cores of $0.15 - 0.45M_\odot$, C/O cores of $0.496 - 1.025M_\odot$, and O/Ne cores of $1.259 - 1.376M_\odot$. The user can also create saved models for essentially any purpose through available controls.

The second way to start a new evolution is to create a pre-main sequence (PMS) model by specifying the mass, M, a uniform composition, a luminosity, and a central temperature, T_c low enough that nuclear burning is inconsequential ($T_c = 9 \times 10^5$ K by default). For a fixed T_c and composition, the total mass depends only on the central density, ρ_c. An initial guess for ρ_c is made by using the $n = 1.5$ polytrope, which is appropriate for a fully convective star, but we do not assume the star is fully convective during the subsequent search for a converged PMS model. Instead, `MESA star` uses the `mlt`, `eos`, and Newton solver from `num` to search for a ρ_c that gives a model of the desired mass. The PMS routine presently creates starting models for $0.02 \leq M/M_\odot \leq 50$. Beyond these limits we find

challenges converging the generated PMS model within the `MESA star` evolutionary loop. For such cases it is currently better to generate a starting model within the acceptable mass range, save it, relax it to a new mass with a specified mass gain or loss, and save that model.

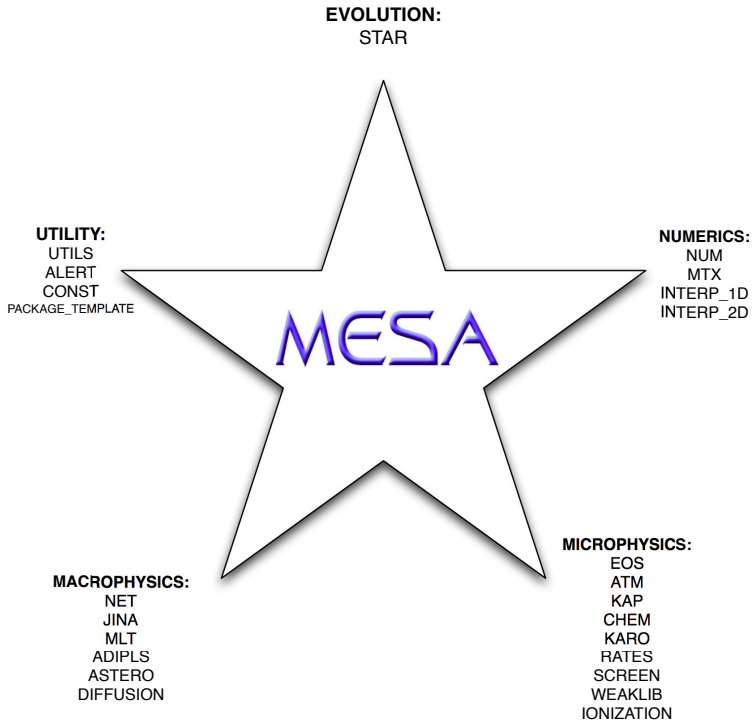

Figure 1.1: Structure of the modules for experiments in stellar astrophysics. This breaks down by purpose the various directories in the MESA folder.

`MESA star` has the ability to create a binary file of its complete current state, called a photo, at user-specified timestep intervals. Restarting from a photo ensures no differences in the ensuing evolution. When restarting from a photo, many controls and options can be changed. A photo is different than a saved model in that a saved model is a text file containing a minimal description of the structure and composition but does not have enough information to allow a perfect restart. However, saved models are not tied to a particular version of the code and therefore are suitable for long term use or sharing with other users.

There are two additional types of output files, logs and profiles. A log records evolutionary properties over time such as stellar age, current mass, and a wide array of other quantities. A profile records model properties at a specified timestep at each zone from surface to center.

WORKING WITH MESA/STAR Make as many copies of the work directory as you'd like, name them anything, and put them wherever you'd like. Edit your make/makefile to set `MESA_DIR` to hold the path to the top level mesa directory. This is what it looks like in Bill's computer:

```
MESA_DIR = /Users/bpaxton/mesa
```

Edit the 'inlist' file in work to redirect to read 'inlist_project'. There are 3 places where you need to have the filename of the inlist:

```
extra_star_job_inlist1_name = 'inlist_project'
extra_controls_inlist1_name = 'inlist_project'
extra_pgstar_inlist1_name = 'inlist_project'
```

Edit the `&star_job` section of the `inlist_project` file to define the path to the mesa data directory. This is what it looks like in Bill's computer:

```
mesa_data_dir = '/Users/bpaxton/mesa/data'
```

Make your work version of star by executing the 'mk' script.

```
./mk
```

First, it compiles 3 files, placing output in your make directory:

```
your src/run_star_extras.f
mesa/star's test/src/run_star.f
your src/run.f
```

Then it links a lot of things to make your 'star' program. Start your copy of mesa star by running the 'rn' script. Remove the objects and the star program by running 'clean'. Change parameters for star by editing inlist_project. The full set of parameters and default values can be found in the star/public directory – see

```
run_star_defaults.dek for the &star_job namelist
star_defaults.dek for the &controls namelist
pgstar_defaults.dek for the &pgstar namelist
```

THE INLIST MESA/star is configured to accept settings and options using fortran namelists. The basic functionality of MESA/star is accessible through two namelists:

1. star_job
2. controls

These can be conveniently combined into one file, which we call an inlist file. There are several additional namelists for specific tasks, such solar model calibration. More on these later. MESA/star has default settings for all of the parameters in star_job and

controls built in to the code. A record of these values is provided in 'inlist_standard' as a reference for users. A number of individual examples are also provided.
To run MESA/star for a particular case, the user needs to create a namelist file that contains the information that the code requires to model that case. The developers highly recommend that the file includes only the minimum number of parameters necessary rather than making a copy of inlist_standard and changing a few parameters. The former approach makes it easier for the user and others to understand what a given inlist is doing and will require a minimum number of adjustments if the settings change from one release to the next.
It is a good idea to browse through inlist_standard in order to become familiar with the variety of options available. The file is quite long but it is broken up into sections. The star_job namelist tells MESA/star what type of calculation it's going to do, what sort of model it's going to start from, and some other information required at startup, such as the nuclear network and opacity tables. The controls namelist tells MESA/star about mass, composition, certain aspects of the physics to be considered, numerical tolerances, timesteps, and when the calculation should end.
When the MESA/star executable is run, it first reads the file 'inlist'. 'inlist' can either include the desired parameter settings or point to another inlist that includes these settings. The developers recommend the latter option because it allows one to keep a record of settings used for a particular model for later reference or sharing. If a parameter isn't specified, the default value (see inlist_standard) is assumed.

2 Advanced MESA

A brief review of MESA organization Bill explains on `mesa.sourceforge.net` that "Each MESA module has its own directory with a standard set of subdirectories and scripts. The standard subdirectories for each module are `make`, `private`, `public`, and `test`. The test directory has `make` and `src` directories for the program that tests the module when it is created. The `public` directory has the sources for the interface to the library, while the `private` directory has sources for the parts of the implementation that are meant for internal use only." See Fig. 2.2 for a scheme of the logical structure of MESA Star.

Adding your code to MESA The idea is that we DO NOT want to edit any file in the MESA folder, neither re-compile the whole code. Ideally we want to modify and compile only something that lives in our MESA Star work directory. This is any directory you created for your runs. It can be anywhere, and only requires a link to the main MESA installation (`MESA_DIR` in `make/makefile`). To do this we need "hooks" to the guts of MESA.

Run_star_extras.f The easiest way to add things is to edit the routines in your private working directory copy of `run_star_extras.f` (in `work/src`). The default version

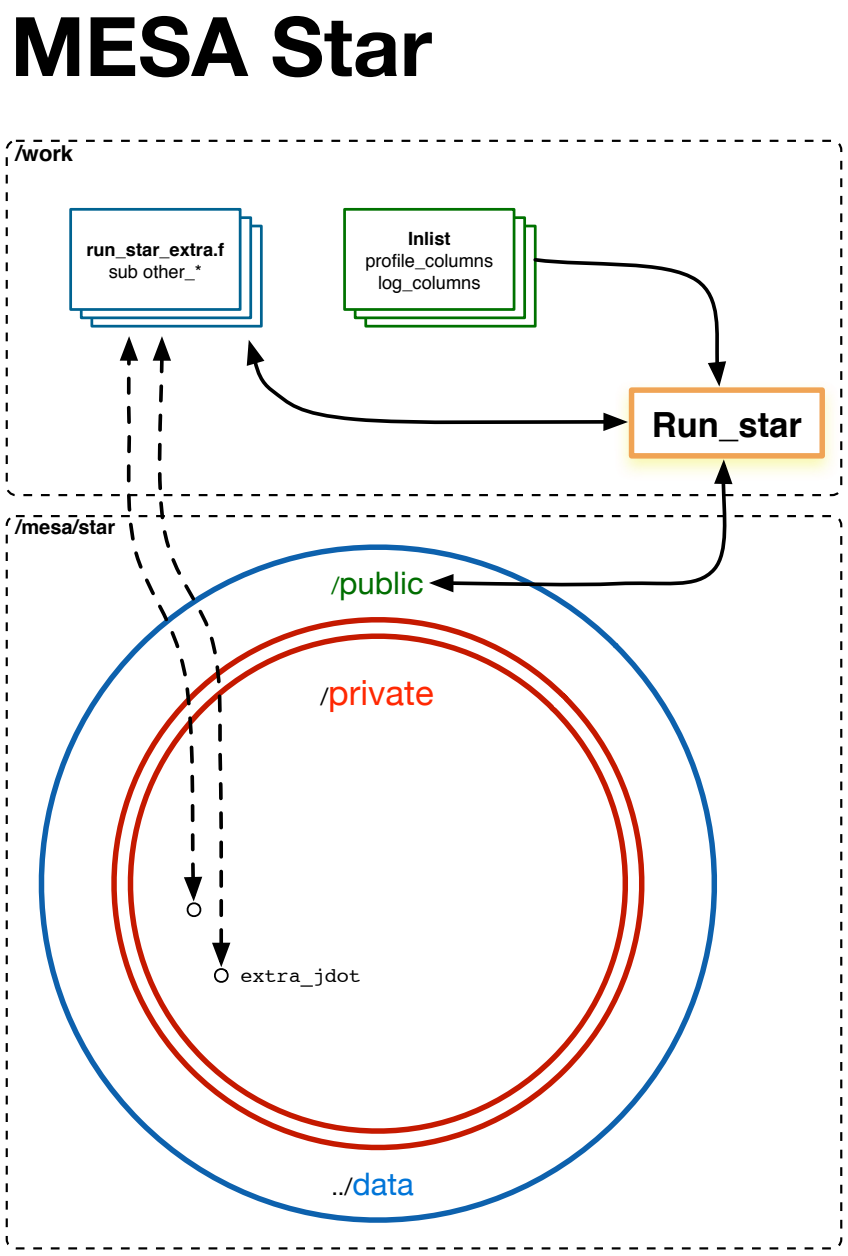

Figure 2.1: MESA Star logical structure

of `run_star_extras` simply includes the file `star/public/standard_run_star_extras.dek`.

1. Replace the "include" line in your `run_star_extras` by the code from the

`standard_run_star_extras.dek` so you can edit your copy. The routines in `run_star_extras` are called from the `do_run_star` routine in `star/test/src/run_star.f`. From those routines, you have access to the star data structures and the `star_lib` routines. Your code can check the state of the model at the end of each step and change controls or save files or whatever else it want to do before the start of the new step.

2. After adding your code, compile (./mk in your work directory) and then run your calculations (./rn).

Useful examples of code that can go into your `run_star_extras` are provided in the `mesa/star/public/` folder.
Two interesting examples are `other_am_mixing.f` and `other_torque.f`.

STAR_INFO The data for a star lives in a derived type called `star_info` that is defined in `public/star_def.f`. You refer to the star either using an integer "id" or a pointer to an instance of the `star_info` type, which is consistently named "s" in the code. There are 2 main sections to the `star_info`: the data, defined in `star_data.dek`, and the controls, defined in `star_controls.dek`. So, for example, if for some reason you want to limit the timestep after the luminosity reaches a certain value, you might do something like this in your `run_star_extras` file:

```
if (s% log_luminosity > 3) &
   s% max_timestep = 1d6*secyer ! limit dt to 1 Myr
```

OTHER_AM_MIXING.F AND OTHER_TORQUE.F These are two empty modules that include a template subroutine ready to be modified and included in your `run_star_extras`. Have a look at these two files and read carefully the commented text.
Using these subroutine you can either directly modify the value of a diffusion coefficient (e.g. Other_am_mixing.f) or use some existing hooks to add/subtract angular momentum from the stellar structure (e.g. Other_torque.f). Of course you wanna know exactly how this is done, so a simple grep of one of these hooks

```
s% extra_jdot
s% extra_omegadot
```

will show a few revealing lines of code in `star/private/solve_omega_mix.f`

```
if (J_tot_extra /= 0) then
           forall (k=1:nz) b(k) = s% omega(k) + &
                  dt*(s% extra_omegadot(k) + s% extra_jdot(k)/s% i_rot(k))
            else
               forall (k=1:nz) b(k) = s% omega(k)
            end if
```

So, just acting from your `run_star_extras` you can act deep down in the guts of MESA, without really having to go there!
Now let's say that you wrote and added your great subroutine `my_am_mixing` in `run_star_extras`. You compile and it works (yey!). You still have one more step to go: you need to add a call to your subroutine in `extras_controls`, which you can find in `run_star_extras` (as you previously copied the content of `standard_run_star_extras.dek` there)

```
subroutine extras_controls(s, ierr)
      type (star_info), pointer :: s
      integer, intent(out) :: ierr
      ierr = 0
      s% other_am_mixing => my_am_mixing
      s% other_torque => my_torque
   end subroutine extras_controls
```

CONTROLS FOR YOUR ROUTINE Your new theory for angular momentum transport has one or more tweak parameters (shame on you!!) and you want to be able to change it from the inlist? Here is some more Bill's wisdom:

```
! NOTE: if you'd like to have some inlist controls for your routine,
! you can use the x_ctrl array of real(dp) variables that is in
! &controls
! e.g., in the &controls inlist, you can set
!     x_ctrl(1) = my_special_param
! then in your routine, you can access that by
!     s% x_ctrl(1)
! of course before you can use s, you need to get it using the id
! argument. Here's an example of how to do that --
! add these lines at the start of your routine:
!         use star_lib, only: star_ptr
!         type (star_info), pointer :: s
!         call star_ptr(id, s, ierr)
!         if (ierr /= 0) then ! OOPS
!            return
!         end if
! for integer control values, you can use x_integer_ctrl
! for logical control values, you can use x_logical_ctrl
```

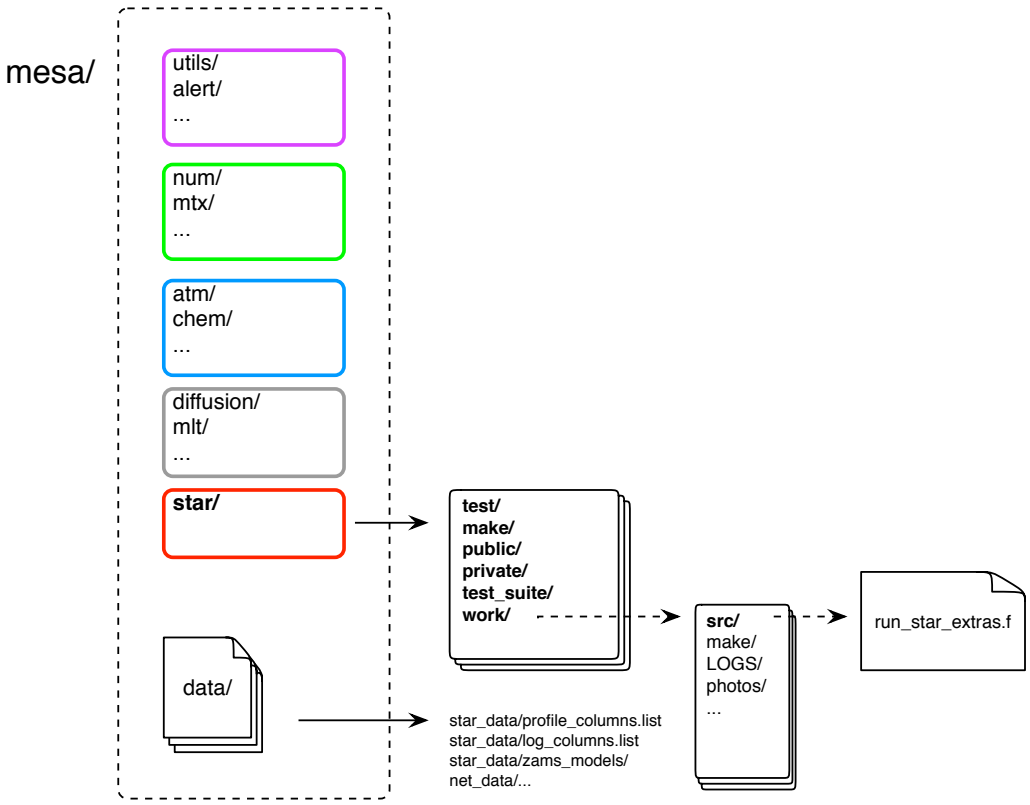

Figure 2.2: Detailed structure of the MESA Star module. The location of the run_star_extras.f file is shown.