How the Parser works.

1 Choosing a model

This can be done either by clicking "select" on the main interface or typing `make <model_name>` on the command line. This sets various quantities in various files, which act like environment variables so that when you carry out the steps below, the parser knows which model is being "made".

2 "Making" the model as a symbolic object

To do this, you either click "make" on the interface or type `make` on the command line.

Making a model as a symbolic object is independent of running simulations. If you have a model which is in some formal way defective, then chances are that this defect will be picked out when "make". The end product of "make" should in most cases be a well formed model. In brief, this is what the parser does (this next part will make more sense if you first read "helpfiles/modules.pdf"):

1. Preliminary activities:
   - Split the descriptor file
   - Create any symbolic links to shared module files if requested in the model descriptor file.
   - Locate all variables and processes defined in the models module files (whether needed by the model or not). Check for repetition and inconsistent definitions
   - Create a single module file containing all modules for convenience
   - Split this single module file into its different components
2. Generate the model as a symbolic object
   - Generate reactions and DEterms from templates
   - Extract the reactions requested in the descriptor file
   - Generate DE terms from the reactions
   - Collect together all DE terms (from reactions, independently defined, etc.)
   - Set any variables chosen to be fixed (default is none – use with care)
   - Generate all the ODEs needed by the model
   - Extract the algrels requested in the model descriptor file
3. Variable and parameter handling (symbolic)
   - Recursively extract all temporary variables needed by the ODEs and algrels.
   - Extract all the key variables and confirm that a rule for their initialisation has been defined. Look through for other unrecognised strings and assume that these are the parameters definitely needed by the model. Confirm that they exist.
   - Recursively get all parameters needed by the model, starting with those which have already been defined

4. Generate code, run simulations, etc.
    - Generate the output variables
    - Generate the parameter setting and variable initialisation functions
    - Compile the parameter setting and variable initialisation functions
    - Generate the constraints
    - Generate the right hand side and mass functions needed by RADAU5
    - Compile the right hand side and mass functions needed by RADAU5
    - Link to produce simulation routine

3 Running simulations

This can be done by clicking "run" on the interface, or by typing `./sim` on the command line. If you are sure that you want to use the default parameter values, input file, and output file for the model, then you can just type `make dat1`.

"make" is performed if necessary. Then parameters are read in, and variables initialised, an input file is read, and the simulation is run using the RADAU5 routine. Output is printed to the file specified.