ShellSet User Guide

Contents

1 Introduction

ShellSet is an MPI parallel combination of three existing programs: OrbData5, Shells and OrbScore2. These programs were originally written, and are currently maintained, by Professor Peter Bird and may be found on his website here[1]. This combination into a single MPI framework allowed parallel test options to be added, section 3.1.1 and 3.1.2; the simplification of user input for the three program units, section 3.1.3; use of a single output file for all models run within a single test, section 5; and additional command line arguments (CLAs) depending on the parallel test option chosen, section 4.2.

Within this guide each individual simulation is referred to as a ”model”, a combination of models as a ”test”. Linked websites within the text are repeated inside footnotes for printed versions.

1.1 Repository content

Table 1 shows an outline of the files and directories found within the ShellSet repository:

Name	Outline
src	Directory containing all Fortran program files
EXAMPLES	Directory containing List & Grid example directories
INPUT	Directory containing all ShellSet, OrbData, Shells and OrbScore input files
Licence.txt	File containing copy of GNU software licence
Makefile	Makefile for 3 compilable versions of ShellSet
README	README file for ShellSet

Table 1: ShellSet repository information

1.2 Prerequisites & environment setup

ShellSet requires an Intel Fortran compiler, MPI and Intel MKL. Each of these is available for free from Intel through the Intel oneAPI Base & HPC toolkits.

Users with Windows machines that do not currently have access to a Linux environment should read the following installation and setup instructions. Users with an existing Linux environment or Linux OS machine can skip ahead to the paragraph related to the installation of the oneAPI toolkits. ShellSet will function in any Linux environment however, for brevity, only the installation instructions for Windows Subsystem for Linux version 2 (WSL2; free from Microsoft) are given. WSL2 has some advantages over a virtual machine including allowing user access to GPU or other accelerators.

The first step is to install WSL2 onto a Windows host, the installation depends on the Windows OS version and build number. For complete instructions follow one of the following two links:

1. for newer versions of Windows follow this link[2],
2. users with older Windows versions should use the following link[3].

The shared folder between the WSL2 environment and Windows host is located here: \\wsl$\Ubuntu-20.04\home, making adjustments for the Linux flavour and version.

[1]http://peterbird.name
[2]https://docs.microsoft.com/en-us/windows/wsl/install
[3]https://docs.microsoft.com/en-us/windows/wsl/install-manual

The next step is the installation of the Intel oneAPI toolkits, instructions for which can be found here[4], a minimum for ShellSet is the installation of the Base & HPC toolkits. For a complete list of all Intel oneAPI toolkits available and their contents see here[5]. The installation instructions will depend on the flavour of Linux and each step must be followed from within the Linux environment. Final setup of the ifort compiler is done by adding the following line to the .bashrc file:
source /opt/intel/oneapi/setvars.sh > /dev/null 2>&1

1.3 Download and unzip

Once a suitable Linux environment is set up it is possible to download the required version of ShellSet and start preparing it for use. If using WSL2 then it is possible that the user will see a new file with the extension ":Zone.Identifier". This is a file created by Windows to mark its companion file as downloaded from an internet source. Removal of this file is not required however, they can clutter a directory and it is safe to remove them, This is possible from the WSL2 terminal, using any typical delete commands, or from the Windows host, see here[6] for details.

Once the file is unzipped (**unzip ShellSet.zip**), the user should navigate inside the newly created directory (**cd ShellSet**) where they can compile the program ready for use, see section 2.

[4]https://www.intel.com/content/www/us/en/develop/documentation/installation-guide-for-intel-oneapi-toolkits-linux/top/installation/install-using-package-managers/apt.html

[5]https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html#gs.8a3psb

[6]https://softwaretested.com/windows/deleting-zone-identifier-files-what-we-know-so-far

2 Compiling

ShellSet comes complete with a makefile in order to simplify its compilation. The standard version of ShellSet may be complied by navigating into the ShellSet directory and entering the following (case sensitive) command:
make ShellSet

Including the basic ShellSet version, the makefile contains three build targets:

1. **ShellSet** build the basic version of ShellSet
2. **debug** build using extra debug flags (currently ”-check all -traceback -debug”)
3. **optimal** build using extra optimising flags (currently ”-xHost -ipo -unroll-aggressive”)

and two clean options:

1. **clean** clean directory and some sub-directories
2. **cleanall** clean directory, remove some sub-directories and all .exe files

Each target creates its own private directory where it stores its dependency files: ”lib” for **ShellSet**; ”lib_DB” for **debug**; and ”lib_OPT” for **optimal**. The makefile allows for all three targets to be built from the same source files without sharing any dependency on compiled files, this means all three targets can be built without any problems from compile flags (such as -ipo), and may be built at the same time using the command ”make ShellSet debug optimal”.

To alter the current debug or optimisation flags users should update the makefile macro DBFLAGS or OPTFLAGS respectively. For a list of compiler options see the Intel alphabetical option list here[7]. Older versions of the ifort compiler require a change to the MKLFLAGS macro in the makefile, from ”-qmkl=parallel” to ”-mkl=parallel”.

The compile line flag ”-diag-disable 10145” stops warning ”ifort: warning #10145: no action performed for file ...”. The linker line flag ”-diag-disable 10182” stops the warning ”ifort: warning #10182: disabling optimization; runtime debug checks enabled”. These may be removed at the users discretion.

[7] https://www.intel.com/content/www/us/en/develop/documentation/fortran-compiler-oneapi-dev-guide-and-reference/top/compiler-reference/compiler-options/alphabetical-option-list.html

3 Program Setup

3.1 ShellSet input

The following subsections explain the input files needed for each model selection option of ShellSet. Although Fortran is not case sensitive in its internal variable names, these input files are. Care must be taken in insert the file name using the correct case, for variable cases see column 1 of table 2. All input files must be saved in the directory INPUT.

3.1.1 List

The simplest of the two model selection options is the direct *List* input choice. This allows the user to input direct values for defined parameters yielding a specific number of simulations. The format for the file "ListInput.in" is shown in listing 1, with an example in listing 2.

```
Number Variables, Number Models
Var 1 Name
Var 2 Name
...
---------- Var1:
Var 1 value 1
Var 1 value 2
...
---------- Var2:
Var 1 value 1
Var 1 value 2
...
---------- VarN:
...
```

Listing 1: Format for the input file "ListInput.in"

```
2,3
fFric
tauMax
---------- Var1:
0.9
0.86
---------- Var2:
1.000E+12
2.000E+12
```

Listing 2: Example "ListInput.in"

This example generates two models using the two variables, model 1 using values 0.9, & 1.000E+12 and model 2 using 0.86 & 2.000E+12.

3.1.2 Grid

If using the grid search option the user should update the file "GridInput.in" with the required information. Listing 3 shows the layout of the file, listing 4 shows an example file.

```
Number Variables, Number Cells kept, Number Search Levels (incl. initial)
----------
Var 1 Name
Var 1 Min Value
Var 1 Max Value
Var 1 Number of models
----------
Var 2 Name
Var 2 Min Value
Var 2 Max Value
Var 2 Number of models
----------
...
```

Listing 3: Format for the input file "GridInput.in"

```
2,2,2
----------
fFric
0.02
0.82
2
----------
tauMax
1.0E12
4.0E12
2
```

Listing 4: Example "GridInput.in"

This example generates 4 models for 2 variables within the defined parameter search area. For the second level the program keeps the best 2 cells from the first, in each of the kept cells another 4 models are generated, for a total of 8 models in level 2 and 12 in total. Further levels would continue in the same fashion. The cells to keep are decided by ranking each model by a misfit score, each model being assumed to represent the entire cell.

ShellSet treats the minimum and maximum values for each variable as "strict" limits that will never be included in the search. Because of this, and so as not to focus the search away from the boundaries, ShellSet shifts the variable values selected towards these set limits. The following is an example using the fFric values given in listing 4. First, ShellSet calculates the distance between any neighbouring models:

$$\text{step size} = \frac{0.82 - 0.02}{2} = 0.4$$

then finds the first model using a half step size:

$$\text{model1} = 0.02 + \frac{0.4}{2} = 0.22$$

and any remaining models using the step size:

$$\text{model2} = \text{model1} + 0.4 = 0.62$$
$$\text{model}\mathbf{N} = \text{model}\mathbf{N\text{-}1} + 0.4$$

In the listing 4 example there are only two models, with fFric values of 0.22 and 0.62.

3.1.3 OrbData, Shells, OrbScore

The input files required for the original program units are not altered, please see Peters website here[8] for information about them. ShellSet reads the names of these files from the input file "InputFiles.in". An example section of this file is shown in listing 5.

```
Shells:
Earth5R.feg               Grid input file (unit 1)
Earth5R-type4AplusA.bcs   Boundary condition file (unit 2)
iEarth5-049.in             Parameter input file (unit 3)
PB2002_plates.dig      Outlines of Plates (unit 8)
PB2002_boundaries.dig     Plate-Pair boundaries (unit 9)
------------------------------
X             Approx Vel solution (unit 11)
X             Mantle Flow (unit 12)
X             Torque & Force balance (unit 13)
X             non-default Lithospheric Rheologies (unit 14)
-------------------------------------------------------------------------
```

Listing 5: Example portion of InputFiles.in showing the Shells program input file list

The first dashed line denotes the boundary between required files and optional files. If an optional filename is given then the program will check for its existence, if the file does not exist then the program continues as if a name were not given. A value of "X" for the optional file will prevent the program from searching for or attempting to use that file. The second, longer, dashed line separates this section of the file from the next. The file contains 4 input list sections, one for each program unit and one for the final run of Shells (outside the main loop) which allows for updated files to be used depending on the target of the work.

For information on the input files see Peter's guide to dynamic modelling of neotectonics with Shells here[9]. One important note is that the first line of the iEarth file will be repeated inside each of the Shells output files (named beginning with fEarth, vEarth & qEarth) on the third line, this line is limited to 100 characters in length, it is advisable to put a small description of the test run as you can see in the iEarth file provided within the ShellSet package. The first and second lines of the output files are again copied from the first line of the .feg and .bcs input files respectively.

3.2 Personalisation & Variable mapping

3.2.1 Variable mapping

Table 2 shows a list of user input variables and their global memory equivalents. The first column lists the variable names that the user will write into the input files (ListInput.in & GridInput.in) in order to tell the program which variables are being updated, the second column shows how the program stores these variables in global memory. A mixture of these variable names must be used when creating a new variable check subroutine, as can be seen in section 3.2.2, depending on whether the subroutine is checking for altered input variables or fixed input variables, care must be taken to reference the correct version of the variable name.

[8] http://peterbird.name
[9] http://peterbird.name/guide/home.htm

User input text	Global memory
fFric	fFric
cFric	cFric
Biot	Biot
Byerly	Byerly
aCreep_C	aCreep(1)
aCreep_M	aCreep(2)
bCreep_C	bCreep(1)
bCreep_M	bCreep(2)
cCreep_C	cCreep(1)
cCreep_M	cCreep(2)
dCreep_C	dCreep(1)
dCreep_M	dCreep(2)
eCreep	eCreep
tAdiab	tAdiab
gradie	gradie
zBAsth	zBAsth
trHMax	trHMax
tauMax	tauMax(1) & tauMax(2)
tauMax_S	tauMax(1)
tauMax_L	tauMax(2)
rhoH2O	rhoH2O
rhoBar_C	rhoBar(1)
rhoBar_M	rhoBar(2)
rhoAst	rhoAst
gMean	gMean
oneKm	oneKm
radius	radius
alphaT_C	alphaT(1)
alphaT_M	alphaT(2)
conduc_C	conduc(1)
conduc_M	conduc(2)
radio_C	radio(1)
radio_M	radio(2)
tSurf	tSurf
temLim_C	temLim(1)
temLim_M	temLim(2)

Table 2: Variable name mapping

3.2.2 Variable Check routines

ShellSet is designed in such that a working thread will call a subroutine to check variable values against specified rules. Currently the file "MOD_VarCheck.f90" contains a single globally valid example subroutine "EarthChk", which checks the values for two pairs of variables. To avoid altering main code regions users should add any localised variable check subroutines to the same module, then add a call from "EarthChk" to any new subroutine(s). Users less proficient in Fortran can use "EarthChk" as a template for new subroutines.

An example section of "EarthChk" is shown in listing 6. This is the first half of a check for a linked pair of variables, namely rhoBar_C and rhoBar_M. The global rule is: **rhoBar_C < rhoBar_M**.

```
if(trim(VarNames(i)) == 'rhoBar_C') then
  tested = .False.
  do j = i+1,size(VarNames)
   if(trim(VarNames(j)) == 'rhoBar_M') then
     tested = .True.
     if(VarValues(j) <= VarValues(i)) then
       Run = .False.
     end if
   end if
  end do

  if(.not. tested) then
    if(rhoBar(2) <= VarValues(i)) then
     Run = .False.
   end if
  end if
...
```

Listing 6: Simplified example of "EarthChk" subroutine, found in MOD_VarChk.f90

Here the subroutine first checks the user input variable names for rhoBar_C, if it finds this then it will check for rhoBar_M. If it finds both then it marks them as *tested* and checks their values against the rule. If it finds only rhoBar_C then *tested* will be *False* and the subroutine will use the global memory value (which is read from the iEarth input file) for rhoBar_M, which from table 2 is rhoBar(2). The second half (not shown) of this section of the check performs the reverse checks, starting from first finding rhoBar_M. Omitted from this simplified section is the error message that is printed to an error file, this error is always non-fatal.

3.3 Misfit Options

OrbScore allows the calculation of up to 6 misfit scores: Geodetic Velocity (GV); Seafloor Spreading Rates (SSR); Stress Direction (SD); Fault Slip Rate (FSR); Smoothed Seismicity Correlation (SC); and Seismic Anisotropy (SA). ShellSet is set up to calculate and output all misfit scores for which the necessary files exist. The following table details example necessary files for each misfit score type. A parameter input file (for example iEarth5-049.in supplied inside folder EXAMPLES) and a Finite Element Grid file (produced by OrbData) are always required.

Smoothed seismic correlation is the only one of these six scores in which a greater value is better, all five other misfit options are preferred with a lower value.

From the table it is clear that the seismic anisotropy option overlaps with the smoothed seismic correlation option, and the seafloor spreading rate overlaps with stress direction. It is not possible to use different files as input for these pairs of misfits as the score of one is connected to the other.

Misfit/Scoring option	Code	Name in EXAMPLES/input
Geodetic Velocity	GV	GPS2006_selected_subset.gps
Stress Direction	SD	robust_interpolated_stress_for_OrbScore2.dat
Seafloor Spreading Rate	SSR	magnetic_PB2002.dat robust_interpolated_stress_for_OrbScore2.dat
Fault Slip Rate	FSR	slip_rate_format.txt
Smoothed Seismic Correlation	SC	GCMT_shallow_m5p7_1977-2017.eqc
Seismic Anisotropy	SA	GCMT_shallow_m5p7_1977-2017.eqc Fouch_2004_SKS_splitting-selected.dat PB2002_plates.dig

Table 3: Files required for each OrbScore2 misfit/scoring options. For a complete list of all IO file options and descriptions see the scoring section of Peter's guide to neotectonic dynamic modelling here[11].

4 Launch & Run

4.1 Terminal command

ShellSet must be launched using two or more MPI threads. If more MPI threads are requested than ever needed then, in order to not waste compute resources, the program will automatically shut down with a warning. The maximum number of threads needed is simply calculated: for the list model option it is the number of models+1; for the grid model selection it is the number of models in the initial level+1.

An example invocation line for the list option is shown here:
mpirun -np 5 ./ShellSet.exe -Iter 3 -InOpt List
This begins a test using five mpi threads, therefore a minimum of four models must be run, and four models will run in parallel. The models are taken from the list model selection option and each will perform three iterations of the "main loop" (see figure 1).

This is an example invocation line for a grid search option:
mpirun -np 5 ./ShellSet.exe -Iter 3 -InOpt Grid -MT GV
This example is the minimum possible since -Iter and -InOpt are required for both list and grid search options. The grid search also needs a misfit score to use in selecting the best cells at each level, in this example the geodetic velocity is selected (-MT GV), for a complete explanation of these and other command line argument options see section 4.2 and table 4.

4.2 Command Line Arguments

Not all command line arguments (CLAs) are required for an invocation and not all are required for each model selection option. Their use can be summarised as: -Iter & -InOpt always required; -Dir, -AEF, -V & -MC optional for both model selection options; -MT, -ML & -KL optional for grid search only. When using the grid search option only one of -MC & -MT need to be defined.

Each CLA is both case and type sensitive. For example -MC expects its first input to be upper case, for the second and third entering a real or integer in place of the other could lead to undefined behaviour.

The majority of the options are quite trivial and explained well in table 4, however some require extra explanation:
-AEF tells ShellSet to stop if model error is detected. This is not recommended but can aid in debugging of a test. Defined Non-fatal errors, such as those generated inside the "EarthChk" subroutine, are never fatal.

[11] http://peterbird.name/guide/Step_22.htm

CLA	Type	Role	Options
-Iter	Integer	Maximum iterations of ”Main Loop” (see figure 1)	User choice
-InOpt	String	Choose between List and Grid input options	”List” or ”Grid”
-Dir	String	Directory where all IO files will be stored	User choice
-AEF		Make all errors fatal to the program	Present means True
-V		Produce extra information in a new output file (see section 5.3)	Present means True
-MC	String Integer Real	To perform misfit convergence: 1) See -MT 2) Iterations of ”Main Loop” before check performed (see figure 1) 3) Multiplied range that defines a ”converged” pair of models.	1) See -MT 2) User choice 3) Range [0,1] e.g: GV,3,0.1
-MT	String	Misfit type used to select best models	User choice: GV, SD, SSR, FSR, SC, SA See table 3
-ML	Real	Create special file to store models with misfit scores better than user defined value (see section 5.2)	User choice
-KL	Real	Misfit score at which Grid Search program will stop	User choice

Table 4: Command Line Arguments (CLAs), their role and user options. All CLAs are case sensitive. All real valued inputs are read with format F8.4 (xxx.yyyy).

-MC requests that ShellSet run misfit convergence. To perform misfit convergence ShellSet generates a misfit score for each iteration of the ”main loop” from the iteration previous to the defined number of iterations at which the check is performed. The next misfit score is compared to the proceeding misfit score by way of comparison using the supplied multiplied range. If the next models misfit score is within the range of the previous models misfit score multiplied by $1 \pm mr$ ($M_n \in [M_{n-1} * (1.0 - mr), M_{n-1} * (1.0 + mr)]$), where mr is the multiplied range, then the two models are considered converged and ShellSet will exit the ”main loop” at this step.

-MT informs ShellSet which misfit score to use to select the best models to keep for future levels in grid search.

-KL supplies a misfit score to ShellSet that, when reached, will stop the program in a controlled way. Should this option be selected, and a misfit score reach the value, ShellSet will print information inside the ”Models.txt” and, if run, the ”Models_Lim.txt” files.

There are also several CLAs which do not begin the main program of ShellSet but instead print only some information, these are referred to as non-running CLAs. Table 5 shows the different options available.

4.3 Errors

There are three different types of error that ShellSet can produce: fatal, always fatal to the entire test set as by definition it is an error that would cause a failure for every model - for example a missing required file; model specific, errors that would cause only that model to fail - for example failing a variable check; and non-fatal errors,

CLA	Role
-help	Print ShellSet help
-info	Print more general ShellSet information
-cite	Print ShellSet citation information

Table 5: Non-running Command Line Arguments (CLAs) and their role. All CLAs are case sensitive.

errors that are not fatal to a model however require reporting - for example not finding an optional file.

Each error type produces its own file containing a message related to the cause of the error. The files are named using the thread ID number of the thread creating the file in order to aid locating a specific error. The files are located inside the "Error" directory which is automatically created upon detection of any type of error. An empty file named for the type of error is also created in the root directory. These empty files are used to stop the program in a controlled way, if a fatal error or model error with flag **-AEF** is found, or with non-fatal errors act as a warning to the user to check the generated message files. The detection of a fatal error will always halt the program, model specific errors will not unless the CLA -AEF (see section 4.2) is used, non-fatal errors will never halt ShellSet.

4.4 Worker Thread

Figure 1 shows a flowchart which outlines the process that a working MPI thread goes through. Diamonds represent decisions, rectangles results and parallelograms IO with the boss thread. Arrows represent the program flow, black for general flow, green and red for true or false responses respectively. As an example the first few steps will be outlined.

1. **Receive Input** - receive several inputs, including the variable values and a logical parameter stating whether to run a model or not. *False* moves the thread to **Exit**, *True* moves the thread to the next step

2. **Input Check** Depending on the variables being altered the thread will run a check of the values using supplied subroutines in MOD_VarCheck.f90 (see section 3.2.2). Failing this moves the thread to **Large Misfit**, passing this moves to the next step.

3. **Run OrbData?** Here the thread checks whether or not to run the program unit OrbData. This is a simple check on which variables are being updated, certain variables require an update to the Finite Element Grid (FEG) file while others do not require any update. If OrbData is not needed then the thread moves to **SHELLS**, otherwise it moves to **OrbData**, after which it moves to **SHELLS** and into the "Main Loop"

The main loop is iterated up to the number of iterations supplied as a CLA (using the option -Iter, see section 4.2), whereupon it will run Shells again with optional altered input files (see section 3.1.3). There are two other ways for the thread to leave the loop: one of the Shells runs fails to converge during its solution step, which would return a large misfit to the boss thread; the misfit convergence option is run which could allow exiting the loop early. The program continues normally even if the convergence criteria are not met within the iterations defined by the user. The final result stored in the output files is always the final run of OrbScore performed on the results from the final run of Shells (outside of the main loop), regardless of any misfit convergence results.

For more information on the individual program units (OrbData, Shells & OrbScore) please see Peter Birds website here[12].

[12]http://peterbird.name

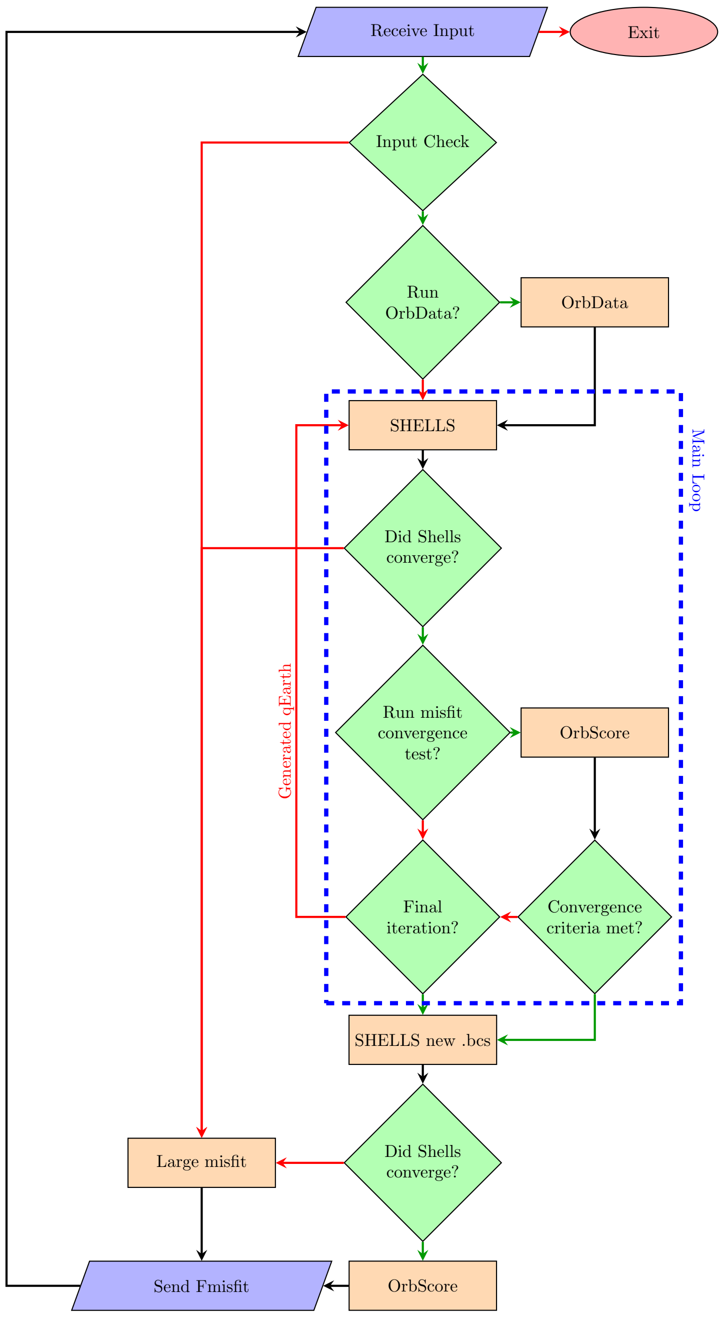

Figure 1: Worker Flow Chart. Yes or True responses in green, No or False responses in red, general flow in black arrows, main work loop in blue dashed box

5 Output

For every test ShellSet will produce one output file, "Models.txt", and can optionally produce two other output files "Models_Lim.txt" and "Verbose**X**.txt" where **X** represents the MPI thread number.

5.1 Models.txt

The test output file is created automatically in the working directory and contains all of the models run within the test, in the order in which they finish. The layout of the file alters slightly depending on the method for model selection. In both cases the first line is used to display the invocation line while the second line shows the order in which the output information is written.

If the list model selector is chosen then the second line will contain the following information: global model number; MPI thread which ran the model; VarValues(**X**), where **X** denotes the number of variables tested; a list of all the misfit scores calculated in the order in which they are printed in the rest of the file. For example, a test of two variables running all misfit options:
global model, ThID, VarValues(2), GV, SSR, SD, FSR, SC, SA

If the grid search model selector is chosen then the second line will contain the following information: global model number; MPI thread which ran the model; grid search level of the model; the cell from the previous level that this model is contained within; VarValues(**X**), where **X** denotes the number of variables tested; a list of all the misfits scores calculated in the order in which they are printed in the rest of the file. For example, a test of two variables running all misfit options:
global model, ThID, Level, Cell, VarValues(2), GV, SSR, SD, FSR, SC, SA
It is important to note that the misfit score chosen to "drive" the grid search is always displayed in the first position.

5.2 Models_Lim.txt

The Models_Lim.txt file is an optional output for the grid search model selector. It's creation is assumed *False* unless switched on at run-time using the **-ML** CLA (see table 4 and section 4.2). If created, the file will contain a list of all models that obtained a misfit score for the driving misfit that is strictly better than its defined value. This file does not impact the creation of Models.txt.

5.3 Verbose.txt

The Verbose**X**.txt, where **X** represents the MPI thread number of the creating thread, is an optional output for either model selector. If switched on, using the **-V** CLA (see table 4 and section 4.2), it will contain all of the information that is typically printed to the terminal by the original three program units: OrbData, Shells and OrbScore. This file is very useful in debugging or learning the specifics of the program units, however for seasoned users can be left off.

5.4 OrbData, Shells, OrbScore

The output files for OrbData, Shells and OrbScore have been renamed compared to the standalone versions. The filenames for Shells and OrbScore are generated by adding the global model number and main loop iteration number to the original file name and unit number, for example the Shells output torque file for the 3rd model number on its 2nd iteration would be saved as: qEarth_3_2.24. OrbData output files are never updated and so they are renamed using only the global model number, for example the same model would produce the OrbData output file: FEG_3.14.

ShellSet organises these files into program unit and thread specific directories within the working directory. For example, Shells output files generated by MPI thread 3 will be stored in the directory: ThID_3_Shells_output.

6 Examples

The folder ”EXAMPLES”, found in the first level of the repository, contains two completed tests, one in the sub-directory List (containing a list model selector test) and the other Grid (containing a grid search model selector test).

The grid search example was run with 2 variables, fFric & tauMax. It was run with 4 models within the fFric variable and 2 models in the tauMax variable direction of the parameter space (8 models per level) over 2 levels, proceeding between levels with the best model, this gives a total of 16 models (8+8). The test was run with 3 ”main loop” iterations for each model. The driving misfit was the geodetic velocity (GV). The invocation line can be seen in the ”Models.txt” file.

The list example shows the same 16 models run as the grid search test. The results between the two example methods are exactly the same.