**MAHLERAN coding practice guide**

To use the versioning software, you need to have a Git client installed. The most straightforward (as it integrates with the GitHub repository) is:

<https://desktop.github.com/>

File 🡪 Clone Repository should allow you to create a local copy you can work on.

The current version of the model is at:

<https://github.com/john-wainwright/Mahleran_v1.2.3>

Use of Git in Netbeans:

<https://netbeans.apache.org/kb/docs/ide/git.html>

The project in Netbeans should look like this:

Graphical user interface

Description automatically generated with medium confidence

And clicking on the arrows next to each subfolder of the Source Files should allow you to see which subroutines are kept where:

Graphical user interface, text, application

Description automatically generated



Text

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

Text

Description automatically generated with low confidence

Text

Description automatically generated with low confidence

Graphical user interface, text, application, chat or text message

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Files with a .for extension are in the old Fortran77 Fixed field format, which has seven spaces on the left before the main code. Only the following should appear in the first seven columns:

c in the first column indicates that the line is a comment.

& in the sixth column (i.e. five spaces then &) shows the line is a continuation from a previous line. Fixed format ignores code after column 72 (because of punch cards!).

Numbers represent either format statements or continuation statements. There should never be the need to use the latter together with goto statements!

Files with a .f90 extension are in free format. Code starts immediately in the first column.

Numbers for format or continuation statements are placed in the first columns. Did I mention never to use goto statements?!

Line length isn’t limited but it’s often a good idea to break longer lines for readability. If doing so, put an & at the *end* of the line to be continued.

Some editors put tabs in the code to space things, but these cause problems when moving between platforms and should be avoided.

To ensure code readability:

* Use spaces liberally:
  + For an array, use x (i, j, k) rather than x(i,j,k)
  + Put spaces around operators: 1 + 2 \* 3 not 1+2\*3
  + Put spaces between function names and arguments:   
    exp (1. + 2. \* sin (y))
  + Put spaces on both sides of an equals sign
  + Put spaces after the commas in a list
* Use variable names that make sense in English. Use an underscore to separate words in a longer variable name.
* Use parentheses to emphasize precedence if you think it could be unclear what the intention of the calculation is. E.g. 1 + (2 \* 3)
* Use alignments to clarify multiple line code:  
  write (ifile, \*) 'Error in input file: specified ', n\_types, &

' surface types, but there should be no more than 10'

* Never have multiple statements on a line (free format allows separation of statements with semi-colons but it makes tracking things down much less clear)

If a value is assigned to a double precision, always write in the decimal point. E.g. 1. + (2. / 3.) and 1 + (2 / 3) will give different results. Some compilers will throw an error if assigned the first of these to an integer variable, but you may find the value is just truncated.