Joint UK Land Environment Simulator (JULES)

Version 3.1

User Manual

Authors: Matt Pryor, Douglas Clark, Phil Harris, Margaret Hendry

Last revised by Matt.Pryor on 22 August 2012.

This version describes JULES v3.1.

Acknowledgements

Thank you to all those JULES users who reported errors or omissions in the previous manual. Hopefully we haven't repeated too many in this version. And thanks in particular for all suggestions for improvements.

1. Introduction and what's new

The Joint UK Land Environment Simulator (JULES) is a computer model that simulates many soil and vegetation processes. This document describes how to run version 3.1 of JULES. It primarily describes the format of the input and output files, and does not include detailed descriptions of the science and representation of the processes in the model.

The first version of JULES was based on the Met Office Surface Exchange System (MOSES), the land surface model used in the Unified Model (UM) of the UK Met Office. After that initial split, the MOSES and JULES code bases evolved separately, but with JULES2.1 these differences were reconciled with the UM.

Further information can be found on the JULES website: http://www.jchmr.org/jules.

1.1. What's new in version 3.1

JULES version 3.1 sees little change to the science of JULES, but contains several major developments intended to make development easier going forward:

1.1.1. Restructuring of the code

The directory structure of the JULES code has been changed to be more logical and allow for a cleaner separation between control, initialisation, I/O and science code. This includes the introduction of directories containing UM-specific code for initialisation in the UM. This was done as part of the work to completely remove JULES code from the UM code repository - it now sits in its own repository.

1.1.2. New I/O framework

The input and output code has been completely revamped in order to modularise and simplify the code. It allows for data to be input on any timestep and interpolated down to the model timestep. Support for outputting of means and accumulations remains. NetCDF is now the only supported binary format (although it should be relatively simple to write drivers for other output formats if desired), and ASCII files are allowed for data at a single location only. Support for the GrADS flat binary format has been dropped, although the NetCDF output should be usable with GrADS with very little work.

1.1.3. User Interface changes

The user interface also sees significant changes. The monolithic run control file has been replaced by several smaller files containing Fortran namelists for input of options and parameters. This is more consistent with the UM, and offers the opportunity to adapt UM tools to provide a GUI for running JULES in the future.

1.1.4. Other changes

There are several not-insignificant changes to the science code:

- Structures are now used for dimensioning variables - this allows for more flexibility of grids than the old system of row_length/rows and halos.

- Move to a new implicit solver - sf_impl2 is now used rather than sf_impl for consistency with the UM. However, the way the implicit coupling is set up means it operates in a similar way to the old scheme.
- A change in the way fresh snow is handled in the multi-layer snow scheme – the density of fresh snow is now prescribed by a new variable (`rho_snow_fresh`). Suggested by Cecile Menard and implemented by Doug Clark.
- Bug fix from Doug Clark for the multi-layer snow scheme that fixes problems with the model oscillating between 0 and 1 snow layers every timestep, preventing snow melt.
- Changes to the sea-ice surface exchange when operating as part of the UM. This will not affect the majority of users.
- Slight changes to the coupling between the explicit and implicit schemes. The vast majority of users will not need to worry about this.

1.2. What's new in version 3.0

The major change in version 3.0 is the introduction of IMOGEN impacts tool. IMOGEN is a system where JULES is gridded on to surface land points, and is forced with an emulation of climate change using "pattern-scaling" calibrated against the Hadley Centre GCM. This climate change - impacts system has the advantage that (a) the pattern-scaling allows estimates of climate change for a broad range of emissions scenarios, (b) new process understanding can be tested for its global implications, (c) new process understanding can also be checked for stability before full inclusion in a GCM and (d) by adding climate change anomalies to datasets such as the CRU dataset, then GCM biases can be removed. It must be recognised that the system is "off-line", and so if major changes to the land surface occur, there might be local and regional feedbacks that can only be predicted using a fully coupled GCM. Hence IMOGEN doesn't replace GCMs, but it does give a very powerful first-look as to potential land surface changes in an anthropogenically forced varying climate.

This was accomplished with help from Mark Lomas at the University of Sheffield and Chris Huntingford at CEH.

There are also several small bug fixes:

- A fix effecting fluxes in sf_stom from Lina Mercado at CEH. This bug fix was announced on the mailing list.
- Small fixes for potential evaporation and canopy snow depth from the UM.
- A small issue with some memory not being deallocated at the end of a run.

1.3. What's new in version 2.2

Along with fixes for known bugs, the changes made for version 2.2 mostly consist of several small additions to the science code. Changes to the control code have mostly been limited to bug-fixes.

- New options for treatment of urban tiles - inclusion of the Met Office Reading Urban Surface Exchange Scheme (MORUSES) and a simple two tile urban scheme.
- Effects of ozone damage on stomata from Stephen Sitch at the University of Leeds.
- New treatment of direct/diffuse radiation in the canopy from Lina Mercado at CEH.

- A new switch allows the competing vegetation portion of TRIFFID to be switched on and off independently of the rest of TRIFFID (i.e. it is now possible to use the RothC soil carbon without having changing vegetation fractions).

There have also been changes made to the way JULES is compiled, due to the re-integration with the Met Office Unified Model. The Unified Model uses pre-processor directives to compile different versions of routines depending on the selected science options. For compatability with this system, JULES will now require a compiler with a pre-processor. This should not be noticed by the majority of users – most modern compilers include a pre-processor and the Makefile deals with setting up the appropriate pre-compiler options.

1.4. What's new in version 2.1?

Version 2.1 of JULES includes extensive modifications to the descriptions of the processes and to the control-level code (such as input and output). These are covered briefly below. Several bug fixes and minor changes to make the code more robust have also been applied. All files are now technically FORTRAN90 (.f90) although many are simply reformatted FORTRAN77 files in which continuation lines are now indicated by the use of the '&' character.

1.4.1. Process descriptions

The main change is that a new multi-layer snow scheme is available (see `nsmax` in Section 5.2). This scheme was developed by Richard Essery at the University of Edinburgh and co-workers. At the time of writing there is little scientific documentation of this development, but this will be made available as soon as possible. In brief, the older, simple scheme represents the snowpack as a single layer with prescribed properties such as density, whereas the new scheme has a variable number of layers according to the depth of snow present, and each layer has prognostic temperature, density, grain size, and solid and liquid water content. The new scheme reverts to the previous, simpler scheme if `nsmax`=0 or when the snowpack becomes very thin.

A four-pool soil carbon model based on the RothC model now replaces the single pool model when dynamic vegetation (TRIFFID) is selected.

There have been several major changes that most users will not notice or need be concerned about. These include a change in the linearization procedure that is used in the calculation of surface energy fluxes (described in the technical documentation). A standard interface is now used to calculate fluxes over land, sea and sea ice. Each surface tile now has an elevation relative to the gridbox mean.

These changes mean that, even with the new snow scheme switched off (`nsmax`=0), results from v2.1 will generally not be identical to those from v2.0.

1.4.2. Control-level code

The major change at v2.1 to the control-level code is that netCDF output is now supported. Both diagnostic and restart files (dumps) can be in netCDF format. There have been several changes to the run control file (see Section 4.6), partly to reflect new science but also in an attempt to organise the file better. These changes mean that run control and restart files from JULES v2.0 are not compatible with v2.1 (although they could be reformatted without too much difficulty).

1.5. What’s new in version 2.0?

The physical processes and their representation in version 2.0 have not changed from version 1. However, version 2.0 is much more flexible in terms of input and output, and allows JULES to be run on a grid of points. New features include:

- Ability to run on a grid.
- Choice of ASCII or binary formats for input and output files (also limited support of netCDF input).
- More flexible surface types – number and types can vary.
- Optional time-varying, prescribed vegetation properties.
- More choice of meteorological input variables.
- Optional automatic spin-up.
- Enhanced diagnostics – large choice of variables, frequency of output, sampling frequency, etc.

2. Overview of JULES

This section provides a brief overview of JULES, largely so as to provide background information and introduce terms used in the rest of the manual. Further details on the science and process descriptions contained in JULES can be found at the JULES website http://www.jchmr.org/jules and in the two JULES description papers (Best et al, 2011, GMD and Clark et al, 2011, GMD).

For both gridded and single point runs, JULES views each gridbox as consisting of a number of surface types. The fractional area of each surface type is either prescribed by the user or modelled by the TRIFFID sub-model. Each surface type is represented by a tile, and a separate energy balance is calculated for each tile. The gridbox average energy balance is found by weighting the values from each tile. In its standard form, JULES recognises nine surface types: broadleaf trees, needleleaf trees, C3 (temperate) grass, C4 (tropical) grass, shrubs, urban, inland water, bare soil and ice. These 9 types are modelled as 9 tiles. A land gridbox is either any mixture of the first 8 surface types, or is land ice. Note that, from version 2.0, one is not limited to these 9 standard surface types (unless running TRIFFID).

Soil processes are modelled in several layers, but all tiles lie over and interact with the same soil column. Each gridbox requires meteorological driving variables (such as air temperature) and variables that describe the soil properties at that location. It is also possible to prescribe certain characteristics of the vegetation, such as Leaf Area Index, to vary between gridboxes.

JULES can be run for any number of gridboxes from one upwards. The number of gridboxes is limited by the availability of computing power and suitable input data. When run on a grid, JULES models the average state of the land surface within the area of the gridbox and most quantities are taken to be homogeneous within the gridbox (with options to include subgrid-scale variability of a few, such as rainfall). In that case, the input data are also area averages. JULES can also be run “at a point”, with inputs that are taken to represent conditions at that point – this configuration might be used when field measurements of meteorological conditions are available.

3. Building and running JULES

Building a JULES executable requires, at the very least, a Fortran 95 compiler with a pre-processor (i.e. any modern Fortran compiler) and a version of the '`make`' utility. The Fortran 90 NetCDF interface library is required to use gridded data (i.e. data for more than a single location). All of this software is freely available:

- GFortran, the GNU GCC Fortran compiler - http://www.gnu.org/software/gcc/fortran/
- GNU make - http://www.gnu.org/software/make/
- NetCDF libraries - http://www.unidata.ucar.edu/software/netcdf/

3.1. Operating system support

JULES has only been tested on Linux but, given a suitable Fortran compiler, should run on any Unix-like system with minimal changes. The recommended way to attempt to run JULES on Windows is via the Linux compatability layer Cygwin (http://www.cygwin.com/), although this is untested.

3.2. Building JULES

JULES executables are built using the `make` utility. This is a piece of software that looks for specific files that tell it how to compile and link together the various files that compose JULES. The `Makefile` supplied with the JULES code should be compliant with most versions of `make`, but is only guaranteed to work with GNU Make.

The JULES `Makefile` uses architecture- and compiler-specific variables that must be set by the user to the appropriate values for their system. Architecture-specific variables, such as the remove command and archiving utility to use, are specified in `Makefile.common.mk`. Compiler-specific variables are given in files named `Makefile.comp.*`. These files are provided for a number of common compilers, e.g. `Makefile.comp.gfortran`. To use a compiler for which a file has not been provided, the user should replace the '`@@`' strings in the blank compiler file `Makefile.comp.misc` with values appropriate to their compiler.

Once the `Makefile` is set up for the user's system, JULES is built by issuing a `make` command at the command prompt while in the directory containing the `Makefile`. The user must specify some options to `make` that will determine how JULES is built:

- **COMPILER:** Determines which version of `Makefile.comp.*` to use for compiler specific options.
- **BUILD:** Allows different compiler flags to be used without changing the `Makefile`s.
- **CDFDUMMY, CDF_LIB_PATH and CDF_MOD_PATH:** Determine how the NetCDF handling code in JULES should be built. This is discussed in more detail below.

The valid values for these options are given in Table 1, and additional information is given in the comments at the head of the `Makefile`.

Table 1 Options that can be passed to `make` when building JULES.

Variable	Permitted values	Notes
`COMPILER`	`sun`	Use options for the Sun Studio compiler series (previously known as Workshop and Forte).
	`intel`	Use options for the Intel Fortran compiler (http://software.intel.com/en-us/articles/fortran-compilers/).
	`g95`	Use options for G95 compiler (http://www.g95.org).
	`gfortran`	Use options for the GNU fortran compiler (http://www.gnu.org/software/gcc/fortran/).
	`nag`	Use options for the NAGWare compiler.
	`pgf`	Use options for the Portland Group compiler.
	`misc`	Use options for an unlisted compiler.
`BUILD`	`run`	**Default option;** Compilation JULES normally.
	`debug`	Switch on compiler debug flags.
	`fast`	Switch on compiler optimisation flags for faster execution.
`CDFDUMMY`	`false`	**Default option;** Use a precompiled netCDF library.
	`true`	Use the dummy netCDF library provided with JULES.

3.2.1. JULES and NetCDF

JULES can be built with or without the NetCDF libraries, however building JULES without NetCDF limits the functionality of JULES.

If the option `CDFDUMMY` is set to `true`, JULES will use a dummy NetCDF library which allows the program to build but provides no functionality. Any attempt to use NetCDF files as input with this option will result in a runtime error. Output files will automatically use a columnar ASCII format with headers. File formats are discussed in more details in Section 4.

If the option `CDFDUMMY` is `false` (or not given), JULES will use the options `CDF_LIB_PATH` and `CDF_MOD_PATH` to determine where to look for the NetCDF library files. `CDF_LIB_PATH` is the directory containing the NetCDF archive files (e.g.. `netcdf.a` and `netcdff.a`). `CDF_MOD_PATH` is the directory containing the NetCDF Fortran 90 module files (e.g. `netcdf.mod`). In a standard NetCDF install, these are often `/usr/lib` and `/usr/include` or `/usr/local/lib` and `/usr/local/include`. If the `nc-config` program is installed on your system (run `which nc-config` to find out), this can be used to determine values for `CDF_LIB_PATH` (`nc-config --flibs`) and `CDF_MOD_PATH` (`nc-config --includedir`). When JULES is built with NetCDF, users can supply either ASCII or NetCDF input files, and all output will be NetCDF.

3.2.2. Example build commands

To create a normal JULES executable without NetCDF using the GFortran compiler:

```
make COMPILER=gfortran BUILD=run CDFDUMMY=true
```

To create a fast JULES executable with the Intel compiler using a NetCDF library:

```
make COMPILER=intel BUILD=fast CDF_LIB_PATH=/path/to/netcdf/lib
CDF_MOD_PATH=/path/to/netcdf/include
```

3.3. Running JULES

The user interface of JULES consists of several files with the extension `.nml` containing Fortran namelists. These files and the namelist members are documented in more detail in Section 5. The JULES executable must be run from the directory containing these namelist files.

For example, to run the Loobos example from a fresh download of JULES:

```
cd /jules/root/dir     (this is the directory containing the Makefile, src etc.)

make COMPILER=gfortran BUILD=run CDFDUMMY=true                  (compiles JULES)

cd examples/point_loobos/

../../jules.exe                                       (runs the JULES executable)
```

4. Input files for JULES

4.1. Overview

The recommended file format for use with JULES is NetCDF, although an ASCII format is also supported *for data at a single location only*. NetCDF is recommended since in this format, the metadata are provided in a standardised manner that many other tools and applications can interpret. The file handling code of JULES is written in a modular way that aims to make it easy for the user to add support for other file formats if they desire (see Section 7.1.3). Any user that does this is strongly encouraged to contribute their code back to the community.

4.2. General principles

JULES supports the input and output of gridded data on both 1D (e.g. vector of land points) and 2D (e.g. latitude/longitude) grids, with an optional additional dimension (e.g. for soil layers). A 2D grid is the usual way to think about gridded data, i.e. with x and y dimensions; however a 1D grid can be more flexible and space-efficient. An example of a 1D grid is a land-points-only grid (as used in the GSWP2 and WATCH datasets). In this case, these data are supplied as a vector of land points, which avoids storing information about sea and sea-ice points that are not being processed. In JULES, the input grid is comprised of the following information:

1. Whether the grid is 1D or 2D (ASCII or NetCDF).
2. The size of each grid dimension (ASCII or NetCDF).
3. The name of each dimension in the file(s) (NetCDF only).

The input grid is specified by the user in the namelist file `model_grid.nml` – see Section 5.5 for more information. *All input data must use the same grid, including any ancillaries and initial conditions.* The model grid is then constructed by selecting the desired points from this input grid, the default being that only the land points in the input grid will be processed. All output is on the model grid.

JULES infers the format of input files from the file extension. The recognised file extensions are:

- **ASCII files**: .asc, .txt and .dat
- **NetCDF files**: .nc and .cdf

4.3. ASCII files

JULES only supports the use of ASCII files for data at a single location. In this case, the input grid can be specified (see Section 4.2) either as a 1D grid with length 1 or as a 2D grid of size 1 x 1. The data should be laid out in columns with one timestep of data per row (with time increasing with the number of rows). For variables with an additional dimension (e.g. soil layers), the values for each level should be in consecutive columns. Variables should be given to JULES in the order they appear in the file, and there should be no unused variables in between. This may mean that some datasets require some preprocessing for use with JULES, even if they are already columnar. If the first character of a line is either `#` or `!`, the line is taken to be a comment. JULES reads no information from comments – they are purely for annotating the dataset for users.

4.3.1. Example ASCII input

ASCII meteorological forcing data

This example is taken from the Loobos data in `loobos/Loobos_1997.dat`:

```
# Meteorological data for Loobos, 1997.
# One year of 30 minute data.
#   Down  Down      Rainfall      Snowfall     Air       Wind   ...
#   SWR   LWR         rate          rate      temp.      speed  ...
#(W m-2) (W m-2)  (kg m-2 s-1) (kg m-2 s-1)    (K)       (m s-1) ...
    0.0  187.8     0.000E+00     0.000E+00   259.800     2.017  ...
    0.0  186.9     0.000E+00     0.000E+00   259.700     3.770  ...
    0.0  186.7     0.000E+00     0.000E+00   259.600     4.290  ...
  ...
```

Each row represents a timestep of data. Each column represents a variable. Driving variables have no additional dimension.

Initial conditions

Taken from the Loobos example – see `examples/point_loobos/initial_conditions.dat`:

```
# sthuf(1:sm_levels)           t_soil(1:sm_levels)
  0.749  0.743  0.754  0.759    276.78  277.46  278.99  282.48
```

Although only one ‘timestep’ of data is supplied, the data must still be laid out in columns. These variables have a value for each soil layer, which are given in consecutive columns. This quickly becomes cumbersome for large numbers of variables, which is why NetCDF is recommended even for data at a single point.

Time varying data with an additional dimension:

```
# lai(1:npft)                  canht(1:npft)
  0.0  0.0  0.2  0.0  0.0       0.0  0.0  0.6  0.0  0.0
  0.0  0.0  0.2  0.0  0.0       0.0  0.0  0.6  0.0  0.0
  0.0  0.0  0.2  0.0  0.0       0.0  0.0  0.7  0.0  0.0
 ...
```

These variables have one value for each plant functional type (see section 2). For each variable, the values for each pft are in consecutive columns. Each row is one timestep of data.

4.4. NetCDF files

For gridded data, NetCDF is the only supported format. Although ASCII files can be used for data at a single location, NetCDF is also the preferred format for such data (due to the reasons discussed in Section 4.1). Files are not expected to use specific dimension or variable names – these are specified via the namelists (see Section 5). The only expectations placed on NetCDF input are:

- All input files use the same grid.
- All input files use the same dimension names (for grid dimensions, additional dimensions and the time dimension).

- The dimensions for each variable appear in the correct order - (points, z, t) for a 1D grid and (x, y, z, t) for a 2D grid, where the z and t dimensions are only present when the variable and context in which the variable is being used require them.
- If using NetCDF for data at a single location, *the grid dimensions are still expected to exist with size 1*.

4.5. File name templating

If the names of input files follow particular patterns, JULES can use a substitution template rather than requiring a potentially long list of file names. Templating comes in two forms, time templating and variable name templating, which can be used separately or together.

Valid substitution strings are listed in Table 2. These are 3-character strings, starting with "%". JULES will automatically detect the use of either form of templating by checking for the presence of the substitution strings in file names.

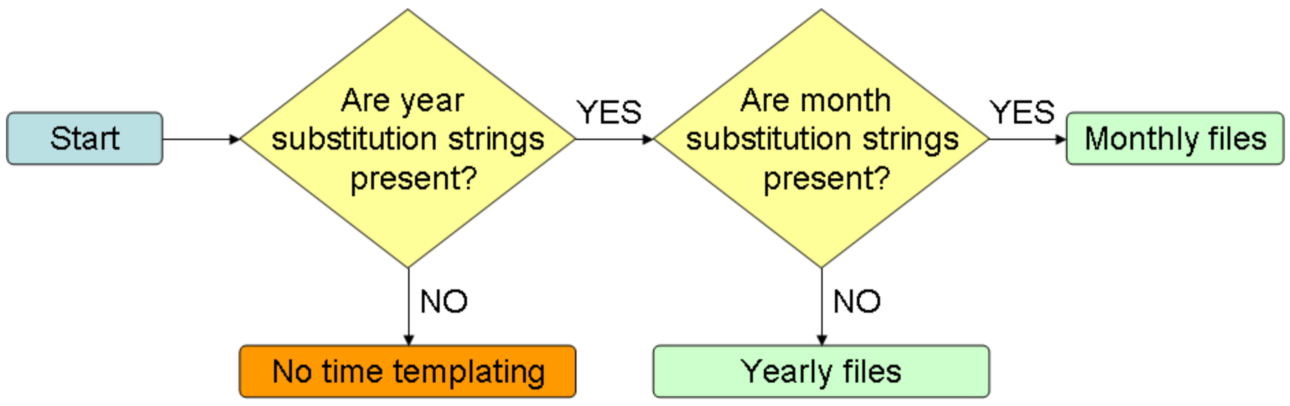
Table 2 Valid substitution strings for substitution templates.

Substitution string	Description
Time templating	
`%y4`	4-digit year
`%y2`	2-digit year
`%m2`	2-digit month
`%m1`	1- or 2-digit month
`%mc`	3-character month abbreviation
Variable name templating	
`%vv`	A character variable

4.5.1. Time templating

If any of the time templating substitution strings are present in a file name, then JULES assumes time-templating is to be used. JULES will automatically detect the period (or frequency) of files based on the specific substitution strings in the following manner:

Figure 1 Flow chart showing process for automatic detection of file period


Start
Are year substitution strings present?
YES
NO
No time templating
Are month substitution strings present?
YES
Monthly files
NO
Yearly files

This means that monthly files must also have a year substitution string present. Only yearly and monthly files are allowed with time templating, with each file containing a single period (year or month respectively) of data. For yearly files, the first data in each file must apply from 00:00:00 on 1st January for each year. For monthly files, the first data in the file must apply from 00:00:00 on the 1st of the month. Other configurations can be specified using a list of files with their respective start times.

4.5.2. Variable name templating

Variable name templating can be used when related variables are stored in separate files with file names that are identical apart from a section that indicates what variable is in each file. Examples of the use of this are given in the next section. JULES will automatically detect if the variable name substitution string is present in a file name, and apply variable name templating if appropriate.

4.5.3. Examples of file name templating

Table 3 Examples of the use of file name templating.

Substitution template	Example file names	Comments

`/data/met_data_%y4%m2.nc`	`/data/met_data_199001.nc`
	`/data/met_data_199002.nc`
	`...`
	`/data/met_data_200410.nc`
		Time templating only.
		Data in monthly files with all related variables in the same file.

`/data/%vv_%y4%mc.nc`	`/data/Rain_1990jan.nc`
	`/data/Wind_1990jan.nc`
	`...`
	`/data/Rain_2000oct.nc`
	`/data/Wind_2000oct.nc`
		Time templating and variable name templating together.
		Data in monthly files with each variable in similarly named but separate files.

`/ancil/soil_%vv.nc`	`/ancil/soil_satcon.nc` `/ancil/soil_sathh.nc`	Variable name templating only. Ancillary (non-time-varying) data with each variable in similarly named but separate files.

`./%vv/met_%vv_199001.nc`
`./%vv/met_%vv_199007.nc`
`...`
`./%vv/met_%vv_199801.nc`
	`./Rain/met_Rain_199001.nc`
	`./Wind/met_Wind_199001.nc`
	`./Rain/met_Rain_199007.nc`
	`./Wind/met_Wind_199007.nc`
	`...`
	`./Rain/met_Rain_199801.nc`
	`./Wind/met_Wind_199801.nc`
		Variable name templating with a list of files (and file start times - not shown).
		Data in 6 monthly files with each variable in similarly named but separate files.
		Since the time templating cannot handle 6 monthly files, the files and their start times must be specified as a list. However, variable name templating can still

		be used. Also note that it is possible to use a substitution string more than once in a template.

4.6. Notes on temporal interpolation

Time-varying input data to JULES require the user to specify how the data should be interpolated onto the model timestep. The permitted interpolation flags are shown in Table 4. These flags are case-sensitive.

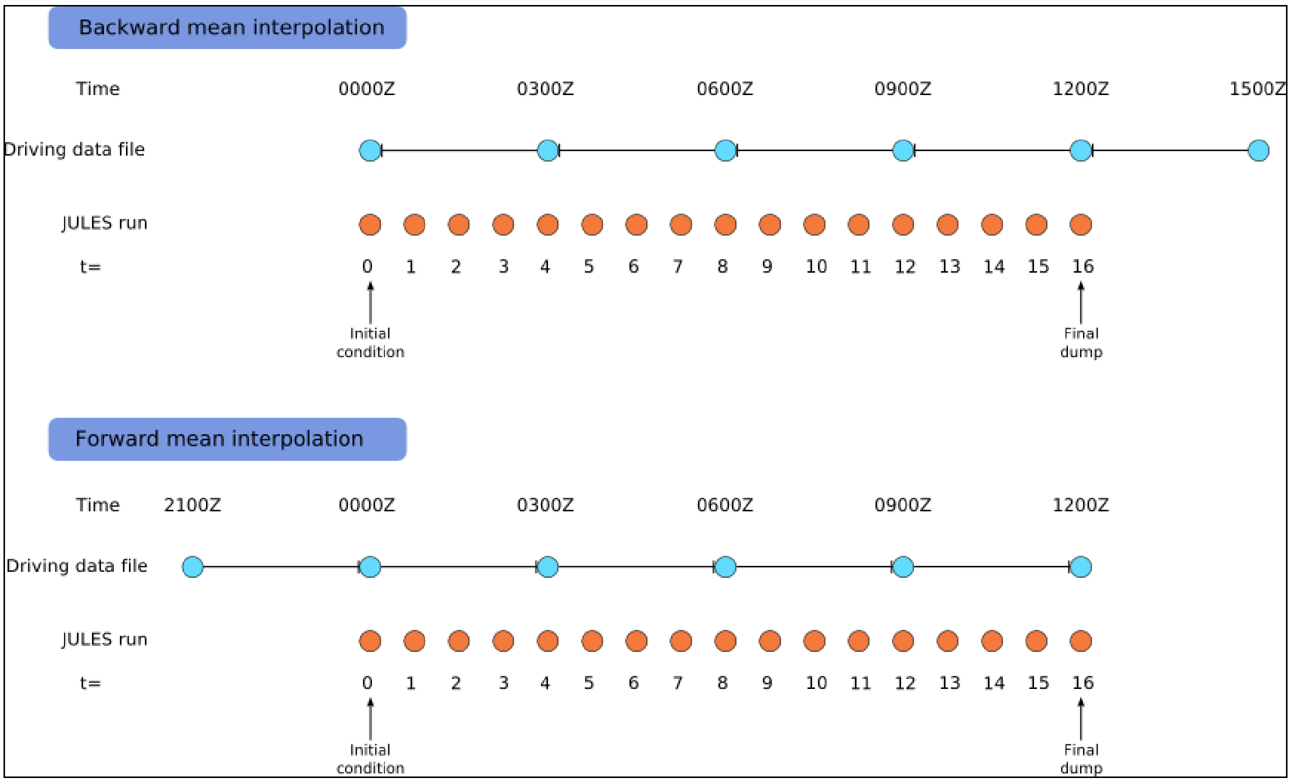
Table 4 Time interpolation flags.

Flag value	Notes
`b`	Backward time average, ending at given time. Will be interpolated with time.
`c`	Centred time average, centred on given time. Will be interpolated with time.
`f`	Forward time average, starting at given time. Will be interpolated with time.
`i`	Instantaneous value at the given time. Will be linearly interpolated with time.
`nb`	Backward time average, ending at given time. Value will be held constant with time.
`nc`	Centred time average, centred on given time. Value will be held constant with time.
`nf`	Forward time average, starting at given time. Value will be held constant with time.

Depending upon the time interpolation flags, driving data may need to be supplied for one or two times that fall before or after the times for the integration. The interpolation scheme implemented in JULES for flags `'b'`, `'c'` and `'f'` is a simplified version of the Sheng and Zwiers (1998)[1] method that conserves the period means of the driving data file. In order to ensure conservation of the average, these flags should be used only if the data period is an even multiple of the model timestep (i.e., if `data_period=2*n*timestep_len`; `n=1, 2, 3`, ...). In these cases the curve-fitting process tends to produce occasional values near turning points that fall outside the range of the input values. Note that for centred data (flags '`c`' and '`nc`') the time of the data should be given as that at the start of the averaging period, rather than the centre. e.g. the 3-hour average over 06H to 09H, centred at 07:30H, should be treated as having timestamp 06H.

[1] Sheng and Zwiers (1998) "An improved scheme for time-dependent boundary conditions in atmospheric general circulation models", *Climate Dynamics*, **14**, 609—613.

Figure 2 Schematic of JULES interpolation of driving variable from a 3 hour timestep to a 45 minute timestep. Simulation start time is 0000Z (on an arbitrary day) and end time is 1200Z. Blue circles indicate driving data required to complete a JULES simulation



5. The JULES namelist files

5.1. Introduction

Each run of JULES is controlled by a number of files containing Fortran namelists. These files specify details including:

- Switches to allow different model configurations to be selected at run-time.
- Start and end times for the run.
- What input data to use and how to read it.
- How to construct the model grid.
- Values for various parameters.
- The required output.

These files have specific names, and JULES expects all these files to exist for every run (even when their contents are not required). The expected files are listed in Table 5, along with a brief description of the options it contains. More detailed descriptions follow in the subsequent sections.

Table 5 namelist files required for a JULES run.

File name	Description	Described in section
`switches.nml`	Switches controlling model behaviour	5.2
`model_levels.nml`	Configuration of the surface types, soil and snow layers.	5.3
`timesteps.nml`	Start and end times for simulation, timestep length and spin-up.	5.4
`model_grid.nml`	Set up the input grid, latitude, longitude, land fractions and model grid.	5.5
`ancillaries.nml`	Gridbox tile fractional coverage, soil properties, PDM and TOMPMODEL parameters and agricultural fraction for use with TRIFFID.	5.6
`pft_params.nml`	Parameters for vegetation tiles.	5.7
`nveg_params.nml`	Parameters for non-vegetation tiles.	5.6.45.8
`triffid_params.nml`	Parameters for TRIFFID dynamic vegetation model.	5.9
`snow_params.nml`	Snow related parameters.	5.10
`misc_params.nml`	Miscellaneous parameters.	5.11
`urban.nml`	Urban model configuration, geometry and material characteristics.	5.12
`imogen.nml`	Options for the IMOGEN analogue model.	5.13
`drive.nml`	Meteorological driving data options.	5.14
`prescribed_data.nml`	Other time-varying input data.	5.15

`initial_conditions.nml`	Initial conditions of prognostic variables.	5.16
`output.nml`	Options for model output.	5.17

5.1.1. Fortran namelists

Each namelist file read by JULES contains one or more Fortran namelists. Any content that does not form part of a namelist group is not read or interpreted in any way by Fortran, and so can be used as comments.

A Fortran namelist combines several related variables (referred to as 'members' of the namelist) together, which are then read with a single statement. The members can appear in any order. A Fortran namelist takes the following format:

```
&GROUP_NAME
  char_variable = "a char variable",

  logical_variable = T,

  nitems = 5,
  list_variable = 0.1  0.2  0.3  0.4  0.5
/
```

The namelist definition is anything that appears between `&GROUP_NAME` and `/`. Values are then declared for the namelist members using the form `member_name = member_value`. The member names are determined by the definition of the namelist in the Fortran source code. The member names for the JULES namelists are documented in the following sections.

Values for character variables must be enclosed in either single(' ') or double (" ") quotes. Logical values should be specified using T for `.TRUE.` or F for `.FALSE.`. Integer and real values are specified simply by giving the value.

Namelists are an ideal input mechanism for programs like JULES that have a large number of inputs, most of which users never change from the default. Since each variable can have a sensible default value specified in the code, the user need only specify variables they wish to change from the default. This can substantially reduce the size of the namelist files. For example, suppose that in the above example the namelist member `logical_variable` has a default value of `.TRUE.`. Then the following namelist specification is equivalent to that above:

```
&GROUP_NAME
  char_variable = "a char variable",

  nitems = 5,
  list_variable = 0.1  0.2  0.3  0.4  0.5
/
```

5.2. `switches.nml`: Switches controlling model behaviour

This file sets up the switches that control model behaviour. It contains a single namelist named `JULES_SWITCHES`.

5.2.1. `JULES_SWITCHES` namelist members

Table 6 Members of the JULES_SWITCHES namelist.

Name	Type	Notes	Default value
`l_aggregate`	logical	Switch controlling number of tiles for each gridbox. This is used to set the number of surface energy balances that are solved for each gridbox (`ntiles`). FALSE: A separate energy balance is calculated for each surface type. This option sets `ntiles=ntype`. TRUE: Aggregate parameter values are used to solve a single energy balance per gridbox. This option sets `ntiles=1`.	F
`can_model`	integer 1 - 4	Choice of canopy model for vegetation: 1: No canopy. 2: Radiative canopy with no heat capacity. 3: Radiative canopy with heat capacity. This option is deprecated, with 4 preferred. 4: As 3 but with a representation of snow beneath the canopy. This option is preferred to 3. NB: `can_model=1` does *not* mean that there is no vegetation canopy. It means that the surface is represented as a single entity, rather than having distinct surface and canopy levels for the purposes of radiative processes.	4

`can_rad_mod`	integer 1 - 5	Switch for treatment of canopy radiation. 1: A single canopy layer for which radiation absorption is calculated using Beer's law. Leaf-level photosynthesis is scaled to the canopy level using the "big leaf" approach. Leaf nitrogen, photosynthetic capacity, i.e the Vcmax parameter, and leaf photosynthesis vary exponentially through the canopy with radiation. 2: Multi-layer approach for radiation interception following the 2-stream approach of Sellers et al. (1992). This approach takes into account leaf angle distribution, zenith angle, and differentiates absorption of direct and diffuse radiation. Leaf-level photosynthesis is calculated using a vertically-varying light-limited rate, and constant Rubisco and export velocities, consistent with the assumption of constant leaf N through the canopy. Canopy photosynthesis and conductance are calculated as the sum over all layers. 3: As 2, but photosynthesis calculated separately for sunlit and shaded leaves for the whole canopy (i.e not at each layer). The definition of sunlit and shaded leaves is based on a threshold of absorbed radiation at each layer. 4: This is a modification of option 2. Instead of constant leaf N through the canopy, it has an exponential decline of leaf N with canopy height. Additionaly includes inhibition of leaf respiration in the light. 5: This is an improvement of option 4, including: • Sunfleck penetration though the canopy. • Division of sunlit and shaded leaves within each canoy level. • A modified version of inhibition of leaf respiration in the light. When using `can_rad_mod=4 or 5`, it is recommended to use driving data that contains direct and diffuse radiation separately rather than a constant diffuse fraction. Descriptions 1, 2 and 3 can be found in Jogireddy et al. (2006), an application of option 4 can be found in Mercado et al. (2007) and all will be described in Clark et al (2011).	4
`ilayers`	integer >= 1	Number of layers for canopy radiation model. Only used if `can_rad_mod` is 2 or 3. These layers are used for the calculations of radiation interception and photosynthesis. A value of 10 is recommended.	10

`l_cosz`	logical	Switch for calculation of solar zenith angle. For land points, this switch is only relevant if `l_spec_albedo=TRUE` (otherwise it is better set to FALSE to prevent unnecessary calculations). TRUE: Calculate zenith angle. FALSE: Assume constant zenith angle of zero, meaning sun is directly overhead.	T
`l_spec_albedo`	logical	Switch for albedo model. TRUE: Use spectral albedo. This includes a prognostic snow albedo. FALSE: Use a single (averaged) waveband albedo.	T
`l_phenol`	logical	Switch for vegetation phenology model. TRUE: Use phenology model. FALSE: Do not use phenology model.	F
`l_triffid`	logical	Switch for dynamic vegetation model (TRIFFID) except for competition. TRUE: Use TRIFFID. In this case soil carbon is modelled using four pools (biomass, humus, decomposable plant material, resistant plant material). FALSE: Do not use TRIFFID. A single sol carbon pool is also used.	F
`l_veg_compete`	logical	Switch for competing vegetation. This is only used if `l_triffid=TRUE`. TRUE: TRIFFID will let the different PFTs compete against each other and modify the vegetation fractions. FALSE: Vegetation fractions do not change.	T
`l_trif_eq`	logical	Switch for equilibrium vegetation model (i.e., an equilibrium solution of TRIFFID). This is only used if `l_triffid=TRUE`. TRUE: Use equilibrium TRIFFID. FALSE: Do not use equilibrium TRIFFID.	F

`l_top`	logical	Switch for a TOPMODEL-type model of runoff production. TRUE: Use a TOPMODEL-type scheme. This is based on Gedney and Cox (2003); see also Clark and Gedney (2008). FALSE: No TOPMODEL scheme. References: Gedney, N. and P.M.Cox, 2003 , *The sensitivity of global climate model simulations to the representation of soil moisture heterogeneity*, J. Hydrometeorology, 4, 1265–1275. Clark and Gedney, 2008, *Representing the effects of subgrid variability of soil moisture on runoff generation in a land surface model*, Journal of Geophysical Research – Atmospheres, 113, D10111, doi:10.1029/2007JD008940.	F
`l_pdm`	logical	Switch for a PDM-type model of runoff production. PDM is the Probability Distributed Model (Moore, 1985), implemented in JULES following Clark and Gedney (2008). TRUE: Use a PDM scheme. FALSE: No PDM scheme. References: Moore, R. J. (1985), *The probability-distributed principle and runoff production at point and basin scales*, Hydrol. Sci. J., 30, 273–297. Clark and Gedney, 2008, *Representing the effects of subgrid variability of soil moisture on runoff generation in a land surface model*, Journal of Geophysical Research – Atmospheres, 113, D10111, doi:10.1029/2007JD008940.	F
`l_anthrop_heat_src`	logical	Switch for inclusion of anthropogenic contribution to the surface heat flux from urban tiles. TRUE: Add anthropogenic effect. FALSE: No effect.	F
`l_o3_damage`	logical	Switch for ozone damage. TRUE: Ozone damage is on. Ozone concentration in ppb must be prescribed in `prescribed_data.nml` (see Section 5.15). FALSE: No effect.	F
`l_imogen`	logical	Switch for IMOGEN TRUE: IMOGEN is used to generate driving data. FALSE: No effect.	F

`l_epot_corr`	logical	TRUE: use correction to the calculation of potential evaporation. FALSE: No effect.	T
`l_snowdep_surf`	logical	TRUE: Use equivalent canopy snow depth for surface calculations on tiles with a snow canopy. FALSE: No effect.	F
`Iscrntdiag`	integer 0 or 1	Switch controlling method for diagnosing screen temperature. 0: Use surface similarity theory. 1: Use surface similarity theory but allow decoupling in very stable conditions based on the quasi-equilibrium radiative solution	0
`l_360`	logical	Switch indicating use of 360 day years. TRUE: Each year consists of 360 days. This is sometimes used for idealised experiments. FALSE: Each year consists of 365 or 366 days.	F
`l_q10`	logical	Switch for use of Q10 approach when calculating soil respiration. TRUE: Use Q10 approach. This is always used if TRIFFID is switched off (`l_triffid=FALSE`) and was used in JULES2.0. FALSE: Use the approach of the RothC model.	T

l_soil_sat_down	logical	Switch for dealing with supersaturated soil layers. If a soil layer becomes supersaturated, the water in excess of saturation will be put into the layer below or above according to this switch. TRUE: Any excess is put into the layer below. Any excess from the bottom layer becomes subsurface runoff. FALSE: Any excess is put into the layer above. Any excess from the top layer becomes surface runoff. This option was used in JULES2.0.	F
l_vg_soil	logical	Switch for van Genuchten soil hydraulic model. TRUE: Use van Genuchten model. FALSE: Use Brooks and Corey model.[2] References: Brooks, R.H. and A.T. Corey, 1964, *Hydraulic properties of porous media.* Colorado State University Hydrology Papers 3. van Genuchten, M.T., 1980, *A Closed-form Equation for Predicting the Hydraulic Conductivity of Unsaturated Soils*. Soil Science Society of America Journal, 44:892-898.	F
soilhc_method	integer 1 or 2	Switch for soil thermal conductivity model.. 1: Use approach of Cox et al (1999), as in JULES2.0. 2: Use approach of Johansen (1975).	1
l_point_data	logical	Flag indicating if driving data are point or area-average values. This affects the treatment of precipitation input and how snow affects the albedo. TRUE: Driving data are point data. Precipitation is not distributed in space (see FALSE below) and is all assumed to be “large-scale” in origin. The albedo formulation is suitable for a point. FALSE: Driving data are area averages. The precipitation inputs are assumed to be exponentially distributed in space, as in UMDP25, and can include convective and large-scale components. The albedo formulation is suitable for a gridbox.	F

[2] In the JULES2.0 User Manual this was described as the “Clapp and Hornberger” model and the JULES code still refers to “Clapp and Hornberger” rather than “Brooks and Corey”. In fact there are important differences between these two hydraulic models (Toby Marthews, pers comm.). There has been confusion in the literature and in past documentation of MOSES/JULES, resulting in these differences often being ignored, but JULES uses the Brooks and Corey model. Hopefully this confusion will be removed from future releases.
Reference: Clapp, R.B. and G.M.Hornberger, 1978, *Empirical Equations for Some Soil Hydraulic Properties*. Water Resources Research 14:601-604.

5.3. `model_levels.nml`: Configuration of the surface types, soil and snow layers

This file configures the surface types, number of soil layers and maximum number of snow layers. It contains a single namelist called `JULES_MODEL_LEVELS`.

5.3.1. `JULES_MODEL_LEVELS` namelist members

Table 7 Members of the JULES_MODEL_LEVELS namelist.

Name	Type	Notes	Default value
`npft`	integer >= 1	The number of plant functional types to be modelled.	5
`nnvg`	integer >= 1	The number of non-plant surface types to be modelled. The total number of surface types to be modelled is called `ntype`, and is given by `ntype=npft+nnvg`. In the standard setup, JULES models 5 vegetation types and 4 non-vegetation types (`npft=5`, `nnvg=4`). However, the model domain need not contain all 9 types – e.g. the domain could consist of a single point with 100% grass. The amount of each type in the domain is set in `ancillaries.nml` (see Section 5.6.1).	4
`urban`	integer	Index of the urban surface type. A negative value indicates no urban surface type.	6
`lake`	integer	Index of the lake surface type. A negative value indicates no lake surface type.	7
`soil`	integer >= 1	Index of the soil surface type. A soil surface type must be given.	8
`ice`	integer	Index of the land-ice surface type. A negative value indicates no land-ice surface type.	9
`urban_canyon`	integer	Index of the urban canyon surface type. A negative value indicates no urban canyon surface type.	-1
`urban_roof`	integer	Index of the urban roof surface type. A negative value indicates no urban roof surface type.	-1

NOTE:	The values of `urban`, `urban_canyon` and `urban_roof` are used to determine what urban scheme to use. • At most one of `urban` and `urban_canyon` can be given. • If `urban_roof` is present, then JULES will use one of the 2-tile urban schemes (and so either `urban` or `urban_canyon` must be present, and will represent the urban canyon surface type). • If `urban_roof` is not present, `urban_canyon` cannot be given and, if present, `urban` is used as the single urban surface type. When giving urban fraction data (see Section 5.6.1), **total** urban fraction should be given in the `urban` or `urban_canyon` tile, whichever is present. This is partitioned internally into roof and canyon.		
`sm_levels`	integer >= 1	Number of soil layers. A value of 4 is often used, and the default soil layer depths have been tuned using this.	4
`nsmax`	integer >= 0	Maximum possible number of snow layers. 0: a composite soil/snow layer is used. This value gives the behaviour found in JULES2.0 and earlier. >0: the state of up to `nsmax` separate snow layers is modelled. Values of `nsmax=3` or more are recommended.	0

5.4. `timesteps.nml`: Start and end times for simulation, timestep length and spin-up

This file sets the start and end time of the run. It can also be used to specify an optional spin-up procedure. It contains two namelists called `JULES_TIME` and `JULES_SPINUP`.

It is recommended that all times use Greenwich Mean Time (GMT or UTC), not local time. The use of GMT is essential if certain options are set (`l_cosz=TRUE`*).*

5.4.1. `JULES_TIME` namelist members

Table 8 Members of the JULES_TIME namelist

Name	Type	Notes	Default value
`timestep_len`	integer >= 1	Model timestep length in seconds. Typically, 30 or 60 minutes is chosen, depending on the driving data available. If the timestep is too long, the model becomes numerically unstable.	None
`main_run_start` `main_run_end`	character	The start and end times for the integration. Each run of JULES consists of an optional spin-up period and the “main run” that follows the spin-up. See below for more about the specification of the spin-up. These variables specify the start and end times for the “main run”. The times must be given in the format **yyyy-mm-dd hh:mm:ss**	None
`phenol_period`	integer >= 1	Period for calls to phenology model (days). Only relevant if `l_phenol` = TRUE.	None
`triffid_period`	integer >= 1	Period for calls to TRIFFID model (days). Only relevant if one of `l_triffid` or `l_trif_eq` is TRUE.	None

5.4.2. `JULES_SPINUP` namelist members

Table 9 Members of the JULES_SPINUP namelist

Name	Type	Notes	Default value
`max_spinup_cycles`	integer >= 0	The maximum number of times the spin-up period is to be repeated: 0: no spin-up >0: at least 1 and at most `max_spinup_cycles` repetitions of spin-up are used. After each repetition, the model tests whether the selected variables have changed by more than a specified amount over the last repetition (see `tolerance` below). If the change is less than this amount, the model is considered to have spun up and the model moves on to the main run.	0
`spinup_start` `spinup_end`	character	Only used if `max_spinup_cycles` > 0. The start and end times for each cycle of spin-up. The times must be given in the format **yyyy-mm-dd hh:mm:ss**	None
`terminate_on_spinup_fail`	logical	Only used if `max_spinup_cycles` > 0. Switch controlling behaviour if the model does not pass the spin-up test after `max_spinup_cycles` of spin-up. TRUE: End the run if model has not spun up. FALSE: Continue the run regardless.	F
Variables used to specify spin-up conditions			
`nvars`	integer >= 0	Only used if `max_spinup_cycles` > 0. The number of variables to use to assess if the model has spun up successfully.	0

`var`	character(nvars)	Only used if `nvars` > 0. List of variables to be used to determine if the model has spun up. Spin-up can be assessed in terms of soil temperature and soil moisture. Possible values are: • smcl: moisture content of each soil layer (kg m^{-2}). • t_soil: temperature of each soil layer (K).	None
`use_percent`	logical(nvars)	Only used if `nvars` > 0. Indicates whether the tolerance for each variable is expressed as a percentage. TRUE: tolerance is a percentage. FALSE: tolerance is an absolute value.	F
`tolerance`	real(nvars)	Only used if `nvars` > 0. Tolerance for spin-up test for each variable. For each spin-up variable, this is the maximum allowed change over a spin-up cycle if the variable is to be considered as spun-up.	None

5.4.3. Note on time conventions

When specifying start times (e.g. `main_run_start`, `spinup_start`), the time is taken to be the *start* of the first timestep. When specifying end times (e.g. `main_run_end`, `spinup_end`), the time is taken to be the *end* of the last timestep. Take the following setup:

```
&JULES_TIME
  timestep_len   = 3600,
  main_run_start = "1997-01-01 00:00:00",
  main_run_end   = "1998-01-01 00:00:00",
...
/
```

With this setup, exactly one whole year of timesteps will be run. The first model timestep begins at 1997-01-01 00:00:00, the second at 1997-01-01 01:00:00 etc. The final model timestep begins at 1997-12-31 23:00:00 and ends at 1998-01-01 00:00:00. No processing occurs for any times in 1998.

5.4.4. Note on solar zenith angle

If a run requires that the solar zenith angle be calculated (`l_cosz` = TRUE), then times must be in Greenwich Mean Time (UTC), so that the code can calculate the zenith angle at each location and time.

If `l_cosz` = FALSE, the user might prefer to use local time, particularly if this is used for input or validation data, as the timestamp on model output will then match that on the other data. However the use of local time is not recommended as if the user later switches to `l_cosz` = TRUE without adjusting the time values, the model results will be in error.

5.4.5. Examples

For all these examples, it is assumed that `l_phenol` = FALSE and `l_triffid` = FALSE, hence we do not need to specify `phenol_period` or `triffid_period` as they will not be used.

A run without spin-up

```
&JULES_TIME
  timestep_len   = 3600,
  main_run_start = "1997-01-01 00:00:00",
  main_run_end   = "1999-01-01 01:00:00"
/

&JULES_SPINUP
  max_spinup_cycles = 0
/
```

This specifies a run with a timestep length of one hour. The run will begin at midnight on 1st January 1997 and end atl 01:00 GMT on 1st January 1999. `max_spinup_cycles = 0` means there is no spin-up.

A run with spin-up over a period that immediately precedes the main run

```
&JULES_TIME
  timestep_len   = 3600,
  main_run_start = "1997-01-01 00:00:00",
  main_run_end   = "1999-01-01 01:00:00"
/

&JULES_SPINUP
  max_spinup_cycles = 5,

  spinup_start = "1996-01-01 00:00:00",
  spinup_end   = "1997-01-01 00:00:00"

  <spinup variable specification>
/
```

This specifies a spin-up period from midnight on 1st January 1996 to midnight on 1st January 1997. This spin-up will be repeated up to 5 times, before the main run from midnight on 1st January 1997 until 01:00 GMT on 1st January 1999.

A run with spin-up over a period that overlaps the main run

```
&JULES_TIME
  timestep_len   = 3600,
  main_run_start = "1997-01-01 00:00:00",
  main_run_end   = "1999-01-01 01:00:00"
/

&JULES_SPINUP
  max_spinup_cycles = 5,

  spinup_start = "1997-01-01 00:00:00",
  spinup_end   = "1998-01-01 00:00:00"

  <spinup variable specification>
/
```

This specifies a spin-up period from midnight on 1st January 1997 to midnight on 1st January 1998. This spin-up will be repeated up to 5 times, before the main run from midnight on 1st January 1997 until 01:00 GMT on 1st January 1999.

Example of specifying requirements for spin-up

```
&JULES_SPINUP
  <spinup time specification>

  terminate_on_spinup_fail = T,

  nvars = 2,
  var         = "smcl"  "t_soil",
  use_percent =     F         T ,
  tolerance   =   1.0       0.1
/
```

With this setup, `terminate_on_spinup_fail` = TRUE means that if the spin-up has not “converged” after `max_spinup_cycles` cycles, the run will end. Convergence is measured using the moisture content and temperature of each soil layer. At every point and in every layer, soil moisture must change by less than 1 kg m^{-2}, while soil temperature must change by less than 0.1%.

5.4.6. Notes on spin-up

Spin-up is assessed using the difference between instantaneous values at the end of consecutive cycles of spin-up. For example, if the spin-up period is from 2005-01-01 00:00:00 to 2006-01-01 00:00:00 then every time the model gets to the end of 2005 the spin-up variables are compared with their value at the end of the previous cycle. The model is considered spun-up when *all* the spin-up variables are spun-up at all points. A spin-up variable is considered spun-up if, at each point, the absolute value of the change (percentage change if `use_percent = TRUE`) over the spin-up cycle is less than or equal to the given `tolerance`.

At present the analysis of whether the model has spun up or not is limited to aspects of the “physical” state of the system, and does not explicitly consider carbon stores, making it less useful for runs with interactive vegetation (the equilibrium mode of TRIFFID is designed to spin-up TRIFFID) or prognostic soil carbon.

During the spin-up phase of a run, JULES provides the correct driving data (for example, meteorological data) as the model time “cycles” round over the spin-up period. Consider the case of a spin-up from 2005-01-01 00:00:00 to 2006-01-01 00:00:00 - at or near the end of 31st December 2005 during the spin-up, the driving data will start to adjust to the values for 1st January 2005. The calculated driving data may vary slightly between the start or end of the first cycle and similar times in later cycles, because of the need to match the data at the end of each cycle to that at the start of the next cycle. When the main run begins after a period of spin-up, the driving data is reset to the start of the main run - no effort is made to adjust the data for a smooth transition. Generally this does not cause a problem.

Depending upon the details of the input data and any temporal interpolation, the driving data may vary rapidly at the end of a cycle of spin-up, causing an extreme response from the model. In most cases the model will adjust, possibly with large heat fluxes over a few hours, but the user should be aware that unusual behaviour near the end/start of a spin-up cycle may be the result of this adjustment. Consider the case of a spin-up from 2005-01-01 00:00:00 to 2006-01-01 00:00:00. At or near the end of 31st December 2005 during the spin-up, the driving data will start to adjust to the values for 1st January 2005, which could be very different from conditions on 31st December 2005. The length of time over which the driving data adjust depends on the frequency of the data, and the choice of temporal interpolation. For example, with 3-hourly data that is interpolated onto a one hour timestep, the adjustment will take place over 3 hours. However, hourly data and an hourly timestep will force an instantaneous adjustment at the start of 1st January 2005.

Although `max_spinup_cycles` specifies the *maximum* number of spin-up cycles, some of which might not be used if the model is considered to have spun up earlier, it is possible to specify the exact number of cycles that will be performed. This can be done by demanding an impossible level of convergence by setting `tolerance`<0 (remember that `tolerance` is compared with the *absolute* change over a cycle) and setting `terminate_on_spinup_fail` to FALSE so that the integration continues when spin-up is judged to have failed after `max_spinup_cycles` cycles.

Although it is expected that a spin-up phase will be followed by the main run in the same integration, it is possible to do the spin-up and main run in separate integrations. This can be done by demanding an impossible level of convergence by setting `tolerance`<0 and setting `terminate_on_spinup_fail` to TRUE so that the integration stops when spin-up is judged to have failed. The final state of the model, after `max_spinup_cycles` cycles of spin-up, will be written to the final dump, and a subsequent simulation started from this dump.

A limitation of the current code is that it cannot cope with a spin-up cycle that is short in comparison to the period of any input data. For example, a spin-up cycle of 1 day cannot use 10-day vegetation data. The code will likely run but the evolution of the vegetation data will probably not be what the user intended! However, it is unlikely that a user would want to try such a run.

Occasionally, the model fails to diagnose a spun up state when in fact the integration has reached a quasi-steady state that is not detected by the procedure of assessing spin-up through comparison of instantaneous values at the end of consecutive cycles of spin-up. An example of this is “period-2” behaviour, where the model state repeats itself over a period of 2 cycles. Such behaviour should be apparent in the model output during spin-up, and the user can opt to repeat the integration over a given number of spin-up cycles, and not to wait for a spun-up state to be diagnosed.

5.5. `model_grid.nml`: Configuration of the input grid and model grid

This file sets up the grid configuration for the run. It contains five namelists - `JULES_INPUT_GRID`, `JULES_LATLON`, `JULES_LAND_FRAC`, `JULES_MODEL_GRID` and `JULES_SURF_HGT`.

Each run of JULES involves two grids: the input grid and the model grid. The input grid is the grid on which *all* input data are held. The model grid is the set of points on which the model is run. The model grid is the grid of points that will be processed by JULES, and is a subset of the input grid.

As discussed in Section 4.2, the input grid consists of three pieces of information:

1. Whether the grid is 1D or 2D.
2. The size of each dimension.
3. The name of each dimension in the input file(s).

The latitude, longitude and land fraction of each point are then read in on the full input grid as specified by the namelists. A subset of the input grid to use as the model grid can then be specified in various ways described below (e.g. land points only, all points within certain latitude/longitude bounds).

In most cases, the model grid will be represented internally as a vector of points, even when the input grid is 2D and the model grid is a rectangular sub-region. The only time that the model grid will be represented by a 2D grid is when the input grid is 2D and the model grid is the whole input grid. Numerically, this makes no difference.

5.5.1. `JULES_INPUT_GRID` namelist members

Table 10 Members of the JULES_INPUT_GRID namelist

Name	Type	Notes	Default value
`grid_is_1d`	logical	Indicates if the input grid is 1D or 2D. TRUE: Variables have one grid dimension in the input file(s) - a points dimensions (e.g. a vector of land points with grid dimension “land”). FALSE: Variables have two grid dimensions in the input file(s) - an x and a y dimension.	F
Only used when `grid_is_1d` = TRUE			
`grid_dim_name`	character	The name of the single grid dimension. For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).	"land"
`npoints`	integer >= 1	The size of the single grid dimension.	0
Only used when `grid_is_1d` = FALSE			
`x_dim_name`	character	The name of the x dimension. For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).	"x"

`y_dim_name`	character	The name of the y dimension. For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).	"y"
`nx`	integer >= 1	The size of the x dimension.	0
`ny`	integer >= 1	The size of the y dimension.	0
Always used			
`pft_dim_name`	character	The dimension name used when variables have an additional dimension of size `npft`. For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).	"pft"
`nvg_dim_name`	character	The dimension name used when variables have an additional dimension of size `nnvg`. For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).	"nvg"
`type_dim_name`	character	The dimension name used when variables have an additional dimension of size `ntype`. For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).	"type"
`tile_dim_name`	character	The dimension name used when variables have an additional dimension of size `ntiles`. For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).	"tile"
`soil_dim_name`	character	The dimension name used when variables have an additional dimension of size `sm_levels`. For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).	"soil"
`snow_dim_name`	character	The dimension name used when variables have an additional dimension of size `nsmax`. For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).	"snow"
`scpool_dim_name`	character	The dimension name used when variables have an additional dimension of size `dim_cs1`. For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).	"scpool"
`time_dim_name`	character	The name of the time dimension in any input files containing time varying data. For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).	"tstep"

5.5.2. JULES_LATLON namelist members

Table 11 Members of the JULES_LATLON namelist

Name	Type	Notes	Default value
Used when input grid consists of a single location (1D and npoints = 1 or 2D and nx = ny = 1)			
`latitude`	real	The latitude of the single grid location.	None
`longitude`	real	The longitude of the single grid location.	None
Used for any input grid with more than a single location			
`file`	character	The name of the file to read latitude and longitude from.	None
`lat_name`	character	The name of the variable containing the latitude data.	None
`lon_name`	character	The name of the variable containing the longitude data.	None

5.5.3. JULES_LAND_FRAC namelist members

Land fraction is the fraction of each gridbox that is land. Currently, JULES considers any gridbox with land fraction > 0 to be 100% land, and all others to be 100% sea (or sea-ice).

Land fraction data can be used to select only land points from the full input grid (see Section 5.5.4).

Table 12 Members of the JULES_LAND_FRAC namelist

Name	Type	Notes	Default value
When the input grid consists of a single location (1D and npoints = 1 or 2D and nx = ny = 1), that single location is assumed to be 100% land. **For any input grid with more than a single location, the following are used**			
`file`	character	The name of the file to read land fraction data from.	None
`land_frac_name`	character	The name of the variable containing the land fraction data.	None

5.5.4. `JULES_MODEL_GRID` namelist members

Members of this namelist are used to select the points to be modelled from the input grid. This can be done in various ways (also see the examples in Sections 5.5.65.5.7 and 5.5.7).

Table 13 Members of the JULES_MODEL_GRID namelist

Name	Type	Notes	Default value
`land_only`	logical	TRUE: Model land points only (from the points that are selected with other options). FALSE: Model all selected points. If `use_subgrid` (see below) is FALSE, the land points will be extracted from the full input grid. If `use_subgrid` is TRUE, the land points will be extracted from the specified subgrid.	T
`use_subgrid`	logical	TRUE: The model grid is a subset of the full input grid, specified using some valid combination of the options below. FALSE: The model grid is the full input grid.	F
Only used if `use_subgrid` is TRUE			
`latlon_region`	logical	TRUE: subset of points to model will be selected using latitude and longitude bounds. FALSE: subset of points to model will be selected using a list of latitude and longitude for each point.	None
Only used if `latlon_region` is TRUE			
`lat_bounds`	real(2)	The lower and upper bounds (in that order) for the latitude to use. The model grid will only contain points where `lat_bounds(1) <= latitude <= lat_bounds(2).`	None
`lon_bounds`	real(2)	The lower and upper bounds (in that order) for the longitude to use. The model grid will only contain points where `lon_bounds(1) <= longitude <= lon_bounds(2).`	None
Only used if `latlon_region` is FALSE			
`npoints`	integer	The number of points to model.	None
`points_file`	character	The name of the file containing the latitude and longitude for each point. Each line in the file should contain the latitude and longitude (in that order) for a point.	None

5.5.5. **JULES_SURF_HGT** namelist members

This namelist sets the elevation of each surface tile, **relative to the gridbox mean elevation**. Note that the gridbox mean elevation is not required anywhere in JULES but is implicit in the near-surface meteorological data that are provided (e.g. higher locations will tend to be colder). The elevation of each tile is used to alter the values of the air temperature and humidity over that tile. All tile elevations must be greater than zero, i.e. tile can only be higher than the gridbox average, because the assumptions used to alter the air temperature and humidity only hold for moving to higher elevations. For many applications, the tile elevation can be set to zero.

Table 14 Members of the JULES_SURF_HGT namelist

Name	Type	Notes	Default value
`zero_height`	logical	Switch used to simplify the initialisation of tile elevation. If `l_aggregate` is TRUE, this switch is also set to TRUE. TRUE: set all tile elevations to zero. This is a very common configuration and is made easier by this switch. FALSE: set all tile heights using data from file.	T
Used if zero_height = FALSE and the input grid consists of a single location			
`surf_hgt_io`	real(ntiles)	The tile elevations for each tile for the single location.	None
Used if zero_height = FALSE and the input grid consists of more than one location			
`file`	character	The name of the file to read tile elevation data from.	None
`surf_hgt_name`	character	The name of the variable containing the tile elevation data.	None

5.5.6. Example - ASCII data at a single location

```
&JULES_INPUT_GRID
  nx = 1,
  ny = 1
/

&JULES_LATLON
  latitude  = 52.168,
  longitude = 5.744
/

&JULES_LAND_FRAC /

&JULES_MODEL_GRID /

&JULES_SURF_HGT
  zero_height = T
/
```

JULES_INPUT_GRID: The default value of `grid_is_1d`, FALSE, is used. This means the user has to specify the extents, `nx` and `ny`, of the input grid. Since all the input data is ASCII, no dimension names are required.

JULES_LATLON: The latitude and longitude of the single location are specified directly in the namelist.

JULES_LAND_FRAC: The land fraction at the single location is assumed to be 100%, so nothing is required.

JULES_MODEL_GRID: Use default options to select the model grid (i.e. land points only from the full input grid). In this case, this leaves the single location as the model grid.

5.5.7. Examples of gridded runs

All the examples in this section assume gridded NetCDF data.

5.5.7.1. Specifying a 1D input grid

In this example, input files contain data on a vector of land points. The land points dimension is called "land". The time dimension for time-varying variables is called "time". The default dimension names are used for all additional dimensions (e.g. pft, tile).

```
&JULES_INPUT_GRID
  grid_is_1D = T,

  npoints = 15238,
  grid_dim_name = "land",

  time_dim_name = "time"
/
```

5.5.7.2. Specifying a 2D input grid

In this example, input files contain data on a 2D latitude/longitude grid. The x dimension is called "lon" and the y dimension is called "lat". The time dimension for time-varying variables is called "time". Variables with an extra tiles dimension use the dimension name "pseudo" for that dimension. All other additional dimensions use their default names.

```
&JULES_INPUT_GRID
  grid_is_1D = F,

  nx = 96,
  ny = 56,

  x_dim_name = "lon",
  y_dim_name = "lat",

  tile_dim_name = "pseudo",

  time_dim_name = "time"
/
```

5.5.7.3. Specifying a subgrid using a given range of latitude and longitude

This can be used with either a 1D or 2D input grid.

```
&JULES_LATLON
  file = 'lat_lon.nc',

  lat_name = 'latitude',
  lon_name = 'longitude'
/

&JULES_LAND_FRAC
  file = 'land_mask.nc',
  land_frac_name = 'land_frac'
/

&JULES_MODEL_GRID
  land_only = F,

  use_subgrid = T,

  latlon_region = T,

  lat_bounds = 55.0  57.0,
  lon_bounds = -5.0  -3.0
/
```

This setup reads latitude, longitude and land fraction for each gridbox in the full input grid (1D or 2D) from the named variables in the specified files.

In `JULES_MODEL_GRID`, `use_subgrid` indicates that a subset of the input grid will be selected as the model grid. `latlon_region` then indicates that latitude and longitude bounds will be used to select the subgrid. `land_only` = FALSE means that sea and sea-ice points will remain in the model grid if any are selected. The model grid will then be a vector containing the selected points (those that fall within the latitude/longitude bounds), even if those points could be used to form a rectangular region.

5.5.7.4. Specifying a subgrid using a list of points

This can be used with either a 1D or 2D input grid.

```
&JULES_LATLON
  file = 'lat_lon.nc',

  lat_name = 'latitude',
  lon_name = 'longitude'
/

&JULES_LAND_FRAC
  file = 'land_mask.nc',

  land_frac_name = 'land_frac'
/

&JULES_MODEL_GRID
  use_subgrid = T,

  latlon_region = F,

  npoints = 4,
  points_file = 'points.txt'
/
```

This setup reads latitude, longitude and land fraction for each gridbox in the full input grid (1D or 2D) from the named variables in the specified files.

In `JULES_MODEL_GRID`, `use_subgrid` indicates that a subset of the input grid will be selected as the model grid. `latlon_region` then indicates that a list of latitudes and longitudes will be used to select the subgrid. `land_only` is not given, meaning it takes its default value, TRUE. This means that any sea or sea-ice points specified in the list of points will be discarded. The model grid will then be a vector containing the selected points (those with the given latitude/longitude).

Assuming that the input grid is a 1 degree grid and the latitude and longitude are given at the centre of the gridbox, `points.txt` should look like this:

```
55.5    -4.5
55.5    -3.5
56.5    -4.5
56.5    -3.5
```

5.5.7.5. The only configuration that yields a 2D model grid

```
&JULES_INPUT_GRID
  grid_is_1d = F,

  nx = 96,
  ny = 56,

...
/

&JULES_LATLON
  <specified from file>
/

&JULES_LAND_FRAC
  <specified from file>
/

&JULES_MODEL_GRID
  land_only = F
/
```

In general, the only configuration that yields a 2D model grid is:

- 2D input grid
- The model grid is the full input grid, including any non-land points

If the input grid is a 2D region where every point is land (i.e. not the whole globe), then `land_only` = TRUE would also yield a 2D model grid.

If any options are set that mean some points from the input grid are not modelled, the model grid will be a vector of points. Computationally, this makes no difference.

5.6. `ancillaries.nml`: Setup of quantities including tile fractional coverage, soil properties

This file sets up the values for tile fractional coverage, soil properties, PDM and TOPMODEL parameters and agricultural fraction. It contains six namelists - `JULES_FRAC`, `JULES_SOIL_PARAM`, `JULES_SOIL_PROPS`, `JULES_PDM`, `JULES_TOP` and `JULES_AGRIC`.

5.6.1. `JULES_FRAC` namelist members

This namelist specifies the fraction of the land area in each gridbox that is covered by each of the surface types. If `l_veg_compete` = TRUE (see Section 5.2), then this information is specified in `initial_conditions.nml` (see Section 5.16). In all other cases, it must be read here.

Note that all land points must be either soil points (indicated by values > 0 of the saturated soil moisture content), or land ice points (indicated by the fractional coverage of the ice surface type [if used] being one). The fractional cover of the ice surface type in each gridbox must be either zero or one – there cannot be partial coverage of ice within a gridbox.

If using either URBAN-2T or MORUSES then the *total* urban fraction should be entered in the `urban_canyon` or `urban` tile, whichever is specified (see Section 5.3). This is partitioned into canyon and roof fractions using the canyon fraction (W/R). The canyon fraction is set in `urban.nml` (see Section 5.12) and can either be prescribed by the user or calculated by an empirical formula described in Section 5.12.1 under `l_urban_empirical`.

Table 15 Members of the JULES_FRAC namelist

Name	Type	Notes	Default value
`file`	character	The name of the file to read surface type fractional coverage data from.	None
`frac_name`	character	The name of the variable containing the surface type fractional coverage data. This is only used for NetCDF files. *For ASCII files, the surface type fractional coverage data is expected to be the first (ideally only) variable in the file.*	None

5.6.2. `JULES_SOIL_PARAM` namelist members

This namelist is responsible for setting soil parameters that are not spacially varying.

Table 16 Members of the JULES_SOIL_PARAM namelist

Name	Type	Notes	Default value
`zsmc`	real > 0	If a depth-averaged soil moisture diagnostic is requested, the average is calculated from the surface to this depth (m).	1.0
`zst`	real > 0	If a depth-averaged soil temperature diagnostic is requested, the average is calculated from the surface to this depth (m).	1.0
`confrac`	real > 0	The fraction of the gridbox covered by convective precipitation.	0.3

`dzsoil_io`	real(sm_levels)	The soil layer depths (m), starting with the uppermost layer. Note that the soil layer depths (and hence the total soil depth) are constant across the domain. In all the examples, JULES uses layer depths of 0.1, 0.25, 0.65 and 2.0m, giving a total depth of 3.0m. It is recommended that this is used unless there is good reason not to.	None

5.6.3. `JULES_SOIL_PROPS` namelist members

This namelist specifies how spacially varying soil properties should be set.

Table 17 Members of the JULES_SOIL_PROPS namelist

Name	Type	Notes	Default value
`const_z`	logical	Switch indicating if soil properties are to be uniform with depth. TRUE: soil characteristics do not vary with depth. FALSE: soil characteristics vary with depth.	F
`file`	character	The file to read soil properties from. If `use_file` (see below) is FALSE for every variable, this will not be used. This file name can use variable name templating (see Section 4.5.2).	None
`nvars`	integer >= 0	The number of soil property variables that will be provided. At present, all variables are required for all runs.	0
`var`	character(nvars)	List of soil variable names as recognised by JULES (see Table 18). Names are case sensitive. *For ASCII files, variable names must be in the order they appear in the file.*	None
`use_file`	logical(nvars)	For each JULES variable specified in `var`, this indicates if it should be read from the specified file or whether a constant value is to be used. TRUE: the variable will be read from the file. FALSE: the variable will be set to a constant value everywhere using `const_val` below.	T
`var_name`	character(nvars)	For each JULES variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data. This is not used for variables where `use_file` = FALSE, but must still be given. *For ASCII files, this is not used - only the order in the file matters, as described above.*	None

tpl_name	character(nvars)	For each JULES variable specified in var, this is the string to substitute into the file name in place of the variable name substitution string. If the file name does not use variable name templating, this is not used.	None
const_val	real(nvars)	For each JULES variable specified in var where use_file = FALSE, this is a constant value that the variable will be set to at every point in every layer. This is not used for variables where use_file = TRUE, but must still be given.	None

If const_z = FALSE, variables read from file must have an additional dimension of size sm_levels (see Section 5.3) using the soil_dim_name given in model_grid.nml (see Section 5.5.1). If const_z = TRUE, variables read from file must have no additional dimension. Some examples of the setup of soil properties can be found in the examples directory.

Table 18 List of soil parameters.

Name	Description
albsoil	Soil albedo. A single (averaged) waveband is used.
b	Exponent in soil hydraulic characteristics.
hcap	Dry heat capacity (J m^{-3} K^{-1})
hcon	Dry thermal conductivity (W m^{-1} K^{-1})
satcon	Hydraulic conductivity at saturation (kg m^{-2} s^{-1})
sathh	If l_vg_soil=TRUE (using van Genuchten model), sathh=1/α (m^{-1}), where α is a parameter of the van Genuchten model. If l_vg_soil=FALSE (using Brooks and Corey model), sathh is the absolute value of the soil matric suction at saturation (m). The suction at saturation is generally less than zero, but JULES uses the absolute value.
sm_crit	Volumetric soil moisture content at the critical point (m^3 water per m^3 soil). The critical point is that at which soil moisture stress starts to restrict transpiration
sm_sat	Volumetric soil moisture content at saturation (m^3 water per m^3 soil). Note that this field is used to distinguish between soil points and land ice points. sm_sat>0 indicates a soil point.
sm_wilt	Volumetric soil moisture content at the wilting point (m^3 water per m^3 soil). The wilting point is that at which soil moisture stress completely prevents transpiration

5.6.4. `JULES_PDM` namelist members

This namelist reads parameter values for the PDM-type parameterisation of surface runoff. The values are only used if `l_pdm` = TRUE. These parameters are held constant across the model domain. For further details of PDM, see references under `l_pdm` in Section 5.2.

Table 19 Members of the JULES_PDM namelist

Name	Type	Notes	Default value
`dz_pdm`	real	The depth of soil considered by PDM (m). A value of ~1m can be used.	1.0
`b_pdm`	real	Shape factor for the pdf.	1.0

5.6.5. `JULES_TOP` namelist members

This namelist reads parameter values for the TOPMODEL-type parameterisation of runoff. The values are only used if `l_top` = TRUE. The description below is very brief. For further details, see the references under `l_top` in Section 5.2.

Table 20 Members of the JULES_TOP namelist

Name	Type	Notes	Default value
`zw_max`	real	The maximum allowed depth to the water table (m). This is the depth to the bottom of an additional layer below the `sm_levels` soil layers and hence should be set to a value greater than `SUM(dzSoil)`. Values of 10 to 15m have been used.	15.0
`ti_max`	real	The maximum possible value of the topographic index. A value of 10 has been used successfully.	10.0
`ti_wetl`	real	A calibration parameter used in the calculation of the wetland fraction. It is used to increment the "critical" value of the topographic index that is used to calculate the saturated fraction of the gridbox. It excludes locations with large values of the topographic index from the wetland fraction. See Gedney and Cox (2003). A value of 2 has been used.	2.0
Members used to set up spacially varying properties			
`file`	character	The file to read TOPMODEL properties from. If `use_file` (see below) is FALSE for every variable, this will not be used. This file name can use variable name templating (see Section 4.5.2).	None
`nvars`	integer >= 0	The number of TOPMODEL variables that will be provided. At present, all variables are required for runs using TOPMODEL.	0

`var`	character(nvars)	List of TOPMODEL variable names as recognised by JULES (see Table 21). Names are case sensitive. *For ASCII files, variable names must be in the order they appear in the file.*	None
`use_file`	logical(nvars)	For each JULES variable specified in `var`, this indicates if it should be read from the specified file or whether a constant value is to be used. TRUE: the variable will be read from the file. FALSE: the variable will be set to a constant value everywhere using `const_val` below.	T
`var_name`	character(nvars)	For each JULES variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data. This is not used for variables where `use_file` = FALSE, but must still be given. *For ASCII files, this is not used - only the order in the file matters, as described above.*	None
`tpl_name`	character(nvars)	For each JULES variable specified in `var`, this is the string to substitute into the file name in place of the variable name substitution string. If the file name does not use variable name templating, this is not used.	None
`const_val`	real(nvars)	For each JULES variable specified in `var` where `use_file` = FALSE, this is a constant value that the variable will be set to at every point. This is not used for variables where `use_file` = TRUE, but must still be given.	None

All of the TOPMODEL variables listed below are expected to have no additional dimension.

Table 21 List of TOPMODEL parameters

Name	**Description**
`fexp`	Decay factor describing how the saturated hydraulic conductivity decreases with depth below the standard soil column (m^{-1}).
`ti_mean`	Mean value of the topographic index in each gridbox.
`ti_sig`	Standard deviation of the topographic index in each gridbox.

5.6.6. `JULES_AGRIC` namelist members

If the TRIFFID vegetation model is used, the fractional area of agricultural land in each gridbox is specified using this namelist. Otherwise, the values in this namelist are not used.

Table 22 Members of the JULES_AGRIC namelist

Name	Type	Notes	Default value
`zero_agric`	logical	Switch used to simplify the initialisation of agricultural fraction. TRUE: set agricultural fraction at all points to 0 FALSE: set spacially varying agricultural fraction using data from file.	T
Used if zero_agric = FALSE and the input grid consists of a single location			
`frac_agr`	real	The agricultural fraction for the single location.	None
Used if zero_agric = FALSE and the input grid consists of more than one location			
`file`	character	The name of the file to read agricultural fraction data from.	None
`agric_name`	character	The name of the variable containing the agricultural fraction data.	None

5.7. `pft_params.nml`: Time- and space-invariant parameters for plant functional types

This file contains a single namelist called `JULES_PFTPARM` that sets time-and space-invariant parameters for plant functional types.

5.7.1. `JULES_PFTPARM` namelist members

This namelist reads the values of parameters for each of the plant functional types (PFTs). These parameters are a function of PFT only. Parameters that also vary with time and location can be prescribed in `prescribed_data.nml` (see Section 5.15). Parameters that are only required if the dynamic vegetation (TRIFFID) or phenology sections are requested are read separately in `triffid_params.nml` (see Section 5.9). Every member must be given a value for every run.

The files given in the `examples` directory are a good starting point for setting up any JULES run, and in the majority of cases can be left untouched.

HCTN24 and 30 refer to Hadley Centre technical notes 24 and 30, available from http://www.metoffice.gov.uk/publications/HCTN.

Table 23 Members of the JULES_PFTPARM namelist

Name	Type	Notes	Default value
`pftname_io`	character(npft)	Name of each PFT. These names are for the user's convenience, and do not have any special significance within JULES.	None
`canht_ft_io`	real(npft)	The height of each PFT (m), also known as the canopy height. The value read here is only used if TRIFFID is not active (`l_triffid = FALSE`). If TRIFFID is active, canopy height is a prognostic variable and its initial value is read in `initial_conditions.nml` (see Section 5.16).	None
`lai_io`	real(npft)	The leaf area index (LAI) of each PFT. The value read here is only used if phenology is not active (`l_phenol = FALSE`). If phenology is active, LAI is a prognostic variable and its initial value is read in `initial_conditions.nml` (see Section 5.16).	None
`c3_io`	integer(npft)	Integer. Flag indicating whether PFT is C3 type. 0: not C3 (i.e. C4). 1: C3.	None
`orient_io`	integer(npft)	Flag indicating leaf angle distribution. 0: spherical. 1: horizontal.	None
`a_wl_io`	real(npft)	Allometric coefficient relating the target woody biomass to the leaf area index (kg carbon m^{-2}).	None
`a_ws_io`	real(npft)	Woody biomass as a multiple of live stem biomass.	None

albsnc_max_io	real(npft)	Snow-covered albedo for large leaf area index. Only used if l_spec_albedo = FALSE. See HCTN30 Eq.2.	None
albsnc_min_io	real(npft)	Snow-covered albedo for zero leaf area index. Only used if l_spec_albedo = FALSE. See HCTN30 Eq.2.	None
albsnf_max_io	real(npft)	Snow-free albedo for large LAI. Only used if l_spec_albedo = FALSE. See HCTN30 Eq.1.	None
alpha_io	real(npft)	Quantum efficiency (mol CO_2 per mol PAR photons).	None
alnir_io	real(npft)	Leaf reflection coefficient for NIR. HCTN30 Table 3.	None
alpar_io	real(npft)	Leaf reflection coefficient for VIS. HCTN30 Table 3.	None
b_wl_io	real(npft)	Allometric exponent relating the target woody biomass to the leaf area index. This is 5/3 in HCTN24 Eq.8.	None
catch0_io	real(npft)	Minimum canopy capacity (kg m^{-2}). This is the minimum amount of water that can be held on the canopy. See HCTN30 p7.	None
dcatch_dlai_io	real(npft)	Rate of change of canopy capacity with LAI (kg m^{-2}). Canopy capacity is calculated as catch0 + dcatch_dlai*lai. See HCTN30 p7.	None
dgl_dm_io	real(npft)	Rate of change of leaf turnover rate with moisture availability.	None
dgl_dt_io	real(npft)	Rate of change of leaf turnover rate with temperature (K^{-1}). This is 9 in HCTN24 Eq.10.	None
dqcrit_io	real(npft)	Critical humidity deficit (kg H_2O / kg air).	None
dz0v_dh_io	real(npft)	Rate of change of vegetation roughness length for momentum with height. Roughness length is calculated as dz0v_dh*canht_ft. See HCTN30 p5.	None
eta_sl_io	real(npft)	Live stemwood coefficient (kg C/m/LAI).	None
fd_io	real(npft)	Scale factor for dark respiration. See HCTN 24 Eq. 56.	None
fsmc_of_io	real(npft)	Moisture availability below which leaves are dropped.	None
f0_io	real(npft)	CI/CA for DQ = 0. See HCTN 24 Eq. 32.	None
g_leaf_0_io	real(npft)	Minimum turnover rate for leaves (/360days).	None

`glmin_io`	real(npft)	Minimum leaf conductance for H_2O (m s^{-1}).	None
`infil_f_io`	real(npft)	Infiltration enhancement factor. The maximum infiltration rate defined by the soil parameters for the whole gridbox may be modified for each PFT to account for tile-dependent factors, such as macro-pores related to vegetation roots. See HCTN30 p14 for full details.	None
`kext_io`	real(npft)	Light extinction coefficient - used with Beer's Law for light absorption through tile canopies. See HCTN30 Eq.3.	None
`kpar_io`	real(npft)	PAR Extinction coefficient (m^2 leaf/m^2 ground).	None
`neff_io`	real(npft)	Scale factor relating V_{cmax} with leaf nitrogen concentration. See HCTN 24 Eq. 51.	None
`nl0_io`	real(npft)	Top leaf nitrogen concentration (kg N/kg C).	None
`nr_nl_io`	real(npft)	Ratio of root nitrogen concentration to leaf nitrogen concentration.	None
`ns_nl_io`	real(npft)	Ratio of stem nitrogen concentration to leaf nitrogen concentration.	None
`omega_io`	real(npft)	Leaf scattering coefficient for PAR.	None
`omnir_io`	real(npft)	Leaf scattering coefficient for NIR.	None
`r_grow_io`	real(npft)	Growth respiration fraction.	None
`rootd_ft_io`	real(npft)	Root depth (m). An exponential distribution with depth is assumed, with e-folding depth `rootd_ft`. Note that this means that generally some of the roots exist at depths greater than `rootd_ft`. See HCTN30 Eq.32.	None
`sigl_io`	real(npft)	Specific density of leaf carbon (kg C/m2 leaf).	None
`tleaf_of_io`	real(npft)	Temperature below which leaves are dropped (K).	None
`tlow_io`	real(npft)	Lower temperature for photosynthesis (deg C).	None
`tupp_io`	real(npft)	Upper temperature for photosynthesis (deg C).	None
`emis_pft_io`	real(npft)	Surface emissivity.	None
`z0hm_pft_io`	real(npft)	Ratio of the roughness length for heat to the roughness length for momentum. This is generally assumed to be 0.1. See HCTN30 p6. Note that this is the *ratio* of the roughness length for heat to that for momentum. It does *not* alter the roughness length for momentum, which is calculated using `canht_ft` and `dz0v_dh` (see above).	None
`fl_o3_ct_io`	real(npft)	Critical flux of O3 to vegetation (nmol/m2/s)	None

`dfp_dcuo_io`	real(npft)	Fractional reduction of photosynthesis with the cumulative uptake of O3 by leaves (/mmol/m2).	None

5.8. `nveg_params.nml`: Parameters for non-vegetation surface types

This file contains a single namelist called `JULES_NVEGPARM` that sets time-invariant parameters for plant functional types.

5.8.1. `JULES_NVEGPARM` namelist members

This namelist reads the values of parameters for each of the non-vegetation surface types. These parameters are a function of surface type only. All parameters must be defined for any configuration.

The files given in the examples directory are a good starting point for setting up any JULES run, and in the majority of cases can be left untouched.

HCTN30 refers to Hadley Centre technical note 30, available from http://www.metoffice.gov.uk/publications/HCTN.

Table 24 Members of the JULES_NVEGPARM namelist

Name	Type	Notes	Default value
`nvgname_io`	character(npft)	Name of each surface type. These names are for the user's convenience, and do not have any special significance within JULES.	None
`albsnc_nvg_io`	real(npft)	Snow-covered albedo. Only used if `l_spec_albedo = FALSE`. See HCTN30 Table 1.	None
`albsnf_nvg_io`	real(npft)	Snow-free albedo. See HCTN30 Table 1. Only used if `l_spec_albedo = FALSE`.	None
`catch_nvg_io`	real(npft)	Capacity for water (kg m^{-2}). See HCTN30 p7	None
`gs_nvg_io`	real(npft)	Surface conductance (m s^{-1}). See HCTN30 p7. Soil conductance is modified by soil moisture according to HCTN30 Eq 35.	None
`infil_nvg_io`	real(npft)	Infiltration enhancement factor. The maximum infiltration rate defined by the soil parameters for the whole gridbox may be modified for each tile to account for tile-dependent factors. See HCTN30 p14.	None
`z0_nvg_io`	real(npft)	Roughness length for momentum (m). See HCTN30 Table 4.	None
`ch_nvg_io`	real(npft)	Heat capacity of this surface type (J K^{-1} m^{-2}). Used only if `can_model` is 3 or 4.	None

`vf_nvg_io`	real(npft)	Fractional coverage of non-vegetation "canopy". Typically set to 0.0, but value of 1.0 used if tile should have a heat capacity in conjunction with `can_model` options 3 or 4.	None
`emis_nvg_io`	real(npft)	Surface emissivity.	None
`z0hm_nvg_io`	real(npft)	Ratio of the roughness length for heat to the roughness length for momentum. This is generally assumed to be 0.1. See HCTN30 p6. Note that this is the *ratio* of the roughness length for heat to that for momentum. It does *not* alter the roughness length for momentum, which is given by `z0_nvg` above.	None

5.9. `triffid_params.nml`: Parameters for the TRIFFID model

This file contains a single namelist called `JULES_TRIFFID` that sets parameters relevant to the TRIFFID submodel.

5.9.1. `JULES_TRIFFID` namelist members

This namelist is used to read PFT parameters that are only needed by the dynamic vegetation model (TRIFFID). Values are not used if TRIFFID is not selected.

Table 25 Members of the JULES_TRIFFID namelist

Name	Type	Notes	Default value
`crop_io`	integer(npft) 0 or 1	Flag indicating whether the PFT is a crop. Only crop PFTs are allowed to grow in the agricultural area. 0: not a crop. 1: a crop.	None
`g_area_io`	real(npft)	Disturbance rate (/360days).	None
`g_grow_io`	real(npft)	Rate of leaf growth (/360days).	None
`g_root_io`	real(npft)	Turnover rate for root biomass (/360days).	None
`g_wood_io`	real(npft)	Turnover rate for woody biomass (/360days)	None
`lai_max_io`	real(npft)	Maximum LAI.	None
`lai_min_io`	real(npft)	Minimum LAI.	None

Note that where a quantity is said to have units of "/360days", this means that it is an amount per 360 days.

5.10. `snow_params.nml`: Snow parameters

This file contains two namelists called `JULES_SNOW_PARAM` and `JULES_RAD_PARAM` that read parameter values that are relevant for snow processes.

5.10.1. `JULES_SNOW_PARAM` namelist members

HCTN30 refers to Hadley Centre technical note 30, available from http://www.metoffice.gov.uk/publications/HCTN.

Table 26 Members of the JULES_SNOW_PARAM namelist

Name	Type	Notes	Default value
`dzsnow_io`	real(nsmax)	Prescribed thickness of each snow layer (m). Only used if `nsmax` > 0. The interpretation of `dzSnow` is slightly complicated and an example of the evolution of the snow layers is given in Table 27. `dzSnow` gives the thickness of each layer when it is not the bottom layer. For the top layer (#1), the minimum thickness is `dzSnow(1)` and the maximum thickness is `2*dzSnow(1)`. For all other layers (iz), the minimum thickness is `dzSnow(iz-1)`, i.e. the given thickness of the previous layer, and the maximum thickness is `2*dzSnow(iz)`, i.e. twice the layer `dzSnow` value, except for the last possible layer (`nsmax`) which has no upper limit. As a snowpack deepens, the bottom layer (closest to the soil; label this as layer b) thickens until it reaches its maximum allowed thickness, at which point it will split into a layer of depth `dzSnow(b)` and a new bottom layer b+1 is added to hold the remaining snow. If a layer becomes thinner than its value in `dzSnow` it is removed and the snow partitioned between the remaining layers. Whenever a layer splits or is removed, the properties of the layer (e.g. temperature) are allocated to the remaining layers. Note that `dzSnow(nsmax)`, the final thickness, is not used but a value must be input.	None

`cansnowpft`	logical(npft)	Flag indicating whether snow can be held under the canopy of each PFT. Only used if `can_model` = 4 (see Section 5.2). The model of snow under the canopy is currently only suitable for coniferous trees. TRUE: snow can be held under the canopy. FALSE: snow cannot be held under the canopy.	F
`rho_snow_const`	real	Constant density of lying snow (kg m-3). This value is used if `nsmax` = 0, in which case all snow is modelled as a single layer of constant density.	250.0
`rho_snow_fresh`	real	Density of fresh snow (kg m-3). This value is only used if `nsmax` > 0.	100.0
`snow_hcon`	real	Thermal conductivity of lying snow (W m^{-1} K^{-1}). See HCTN30 Eq.42.	0.265
`snow_hcap`	real	Thermal capacity of lying snow (J K^{-1} m^{-3}).	0.63e6
`snowliqcap`	real	Liquid water holding capacity of lying snow, as a fraction of snow mass. Only used if `nsmax` > 0.	0.05
`snowinterceptfact`	real	Constant in relationship between mass of intercepted snow and snowfall rate. Only used if `can_model` = 4.	0.7
`snowloadlai`	real	Ratio of maximum canopy snow load to leaf area index (kg m^{-2}). Only used if `can_model` = 4.	4.4
`snowunloadfact`	real	Constant in relationship between canopy snow unloading and canopy snow melt rate. Only used if `can_model` = 4.	0.4

Table 27 gives an example of how the number and thickness of snow layers varies with total snow depth for the case of `nsmax` = 3 and `dzSnow` = (0.1, 0.15, 0.2). Note that if the values given by the user for `dzSnow` are a decreasing series with `dzSnow(i)<=2*dzSnow(i-1)`, the algorithm will result in layers i and i+1 being added at the same time. Don't panic - this should not be a problem for the simulation.

Table 27 An example of the evolution of snow layer thickness.

Snow depth (m)	Number of layers	Layer thickness, uppermost layer first (m)	Comments
<0.1	0		While the depth of snow is less than `dzSnow(1)`, the layer model is not active and snow and soil are combined in a composite layer.
0.1 to <0.2	1	Total snow depth.	The single layer grows until it is twice as thick as `dzSnow(1)`.

0.2 to <0.4	2	0.1,remainder	Above 0.2m, the single layer splits into a top layer of 0.1m and the remaining snow in the bottom layer.
≥0.40	3	0.1,0.15,remainder	At 0.4m depth, layer 2 [which has grown to 0.3m thick, i.e. `2*dzSnow(2)`], splits into a layer of 0.15m and a new bottom layer holding the the remaining 0.15m. As all layers are now in use, any subsequent deepening of the pack is dealt with by increasing the thickness in this bottom layer.

5.10.2. `JULES_RAD_PARAM` namelist members

HCTN30 refers to Hadley Centre technical note 30, available from http://www.metoffice.gov.uk/publications/HCTN.

Table 28 Members of the JULES_RAD_PARAM namelist

Name	Type	Notes	Default value
`r0`	real	Grain size for fresh snow (µm). See HCTN30 Eq.15. Only used if `l_spec_albedo` = TRUE.	50.0
`rmax`	real	Maximum snow grain size (µm). See HCTN30 p4. Only used if `l_spec_albedo` = TRUE.	2000.0
`snow_ggr`	real(3)	Snow grain area growth rates ($\mu m^2 s^{-1}$). Only used if `l_spec_albedo` = TRUE. See HCTN30 Eq.16. The 3 values are for melting snow, cold fresh snow and cold aged snow respectively.	0.6, 0.06, 0.23e6
`amax`	real	Maximum albedo for fresh snow. Only used if `l_spec_albedo` = TRUE. Values 1 and 2 are for VIS and NIR wavebands respectively.	0.98, 0.7
`maskd`	real	Used in exponent of equation weighting snow-covered and snow-free albedo.	50.0
`dtland`	real	Degrees Celsius below zero at which snow albedo equals cold deep snow albedo. This is 2·0 in HCTN30 Eq4. Only used if `l_spec_albedo` = FALSE.	2.0
`kland_numerator`	real	Used in snow-ageing effect on albedo. `kland` is computed by dividing this value by `dtland` - see HCTN30 Eq4. Only used if `l_spec_albedo` = FALSE. Must not be zero.	0.3

5.11. `misc_params.nml`: Miscellaneous surface, carbon and vegetation parameters

This file contains five namelists called `JULES_SURF_PARAM`, `JULES_CSMIN`, `JULES_AERO`, `JULES_SEED` and `JULES_SIGM` that are used to set various parameters not set elsewhere.

Thoughout this section, HCTN24 and 30 refer to Hadley Centre technical notes 24 and 30, available from http://www.metoffice.gov.uk/publications/HCTN.

5.11.1. `JULES_SURF_PARAM` namelist members

Table 29 Members of the JULES_SURF_PARAM namelist

Name	Type	Notes	Default value
`hleaf`	real	Specific heat capacity of leaves (J K^{-1} per kg carbon). See HCTN30 p6.	5.7e4
`hwood`	real	Specific heat capacity of wood (J K^{-1} per kg carbon). See HCTN30 p6.	1.1e4
`beta1`	real	Coupling coefficient for co-limitation in photosynthesis model. See Cox et al. (1999), Eq.61	0.83
`beta2`	real	Coupling coefficient for co-limitation in photosynthesis model. See Cox et al. (1999), Eq.62.	0.93
`fwe_c3`	real	Constant in expression for limitation of photosynthesis by transport of products, for C3 plants. See Cox et al. (1999) Eq.60.	0.5
`fwe_c4`	real	Constant in expression for limitation of photosynthesis by transport of products, for C4 plants. See Cox et al. (1999) Eq.60.	20000.0
`q10_leaf`	real	Q10 factor for plant respiration. See Cox et al. (1999) Eq.66	2.0
`kaps`	real	Specific soil respiration rate at 25 degC and optimum soil moisture (s^{-1}). Only used if not using TRIFFID (`l_triffid` = FALSE). See HCTN24 Eq.16.	0.5e-8
`kaps_roth`	real(4)	Specific soil respiration rate for the RothC submodel for each soil carbon pool. Only used if using the TRIFFID vegetation model (`l_triffid` = TRUE), in which case soil carbon is modelled using four pools (biomass, humus, decomposable plant material, resistant plant material).	3.22e-7, 9.65e-9, 2.12e-8, 6.43e-10

`q10_soil`	real	Q10 factor for soil respiration. Only used if `l_q10` = TRUE. See HCTN24 Eq.17	2.0

5.11.2. `JULES_CSMIN` namelist members

Table 30 Members of the JULES_CSMIN namelist

Name	Type	Notes	Default value
`cs_min`	real	Minimum allowed soil carbon (kg m^{-2}).	1.0e-6

5.11.3. `JULES_AERO` namelist members

Table 31 Members of the JULES_AERO namelist

Name	Type	Notes	Default value
`co2_mmr`	real	Concentration of atmospheric CO2, expressed as a mass mixing ratio.	5.241e-4

5.11.4. `JULES_SEED` namelist members

Table 32 Members of the JULES_SEED namelist

Name	Type	Notes	Default value
`frac_min`	real	Minimum fraction that a PFT is allowed to cover if TRIFFID is used.	1.0e-6
`frac_seed`	real	Seed fraction for TRIFFID.	0.01

5.11.5. `JULES_SIGM` namelist members

Table 33 Members of the JULES_SIGM namelist

Name	Type	Notes	Default value
`pow`	real	Power in sigmodial function used to get competition coefficients. See HCTN24 Eq.3.	20.0

5.12. `urban.nml`: Urban model configuration, geometry & material characteristics

This file contains three namelists called `URBAN_SWITCHES`, `URBAN2T_PARAM` and `URBAN_PROPERTIES`.

This section reads in model configuration choices, geometry & material characteristics data for the urban schemes URBAN-2T and MORUSES. Both these schemes must have an `urban_roof` tile and an `urban_canyon` tile. Values from this section are only used if either of the two-tile urban schemes are enabled by specifying an `urban_roof` tile (see Section 5.3). For parameters that MORUSES does not parameterise, and for any MORUSES parametrisations that are turned off (see Table 34), values from `nveg_params.nml` (see Section 5.8) will be used. Further information on MORUSES, including references, can be found in the technical documentation and under `l_moruses` below.

Table 34 Parameters that may be used from `nveg_params.nml` for the `urban_roof` and `urban_canyon` tile types depending on MORUSES switch configuration. Any non-vegetation parameters not referenced in this table are always used from `nveg_params.nml`.

MORUSES switch	Tile type	TRUE	FALSE
`l_moruses_albedo` (`l_cosz`)	`urban_canyon`	MORUSES	`albsnf_nvg`, `albsnc_nvg`
	`urban_roof`	`albsnf_nvg`, `albsnc_nvg`	
`l_moruses_emissivity`	`urban_canyon`	MORUSES	`emis_nvg`
	`urban_roof`	`emis_nvg`	
`l_moruses_rough`	`urban_canyon` `urban_roof`	MORUSES	`z0_nvg, z0hm_nvg`
`l_moruses_storage`	`urban_canyon`	MORUSES	`ch_nvg, vf_nvg`

5.12.1. `URBAN_SWITCHES` namelist members

Table 35 Members of the URBAN_SWITCHES namelist

Name	Type	Notes	Default value
`l_moruses`	logical	Not used unless an `urban_roof` tile is present (see Section 5.3). Switch for turning on MORUSES. TRUE: use MORUSES parametrisations. FALSE: do not use MORUSES parametrisations. Use urban tile parameters, set in `nveg_params.nml` (see Section 5.8), instead. References: Porson, A., et al. (2010), *Implementation of a new urban energy budget scheme in the MetUM. Part I: Description and idealized simulations*, Quarterly Journal of the Royal Meteorological Society, 136: 1514–1529. doi: 10.1002/qj.668 Porson, A., et al. (2010), *Implementation of a new urban energy budget scheme into MetUM. Part II: Validation against observations and model Intercomparison*, Quarterly Journal of the Royal Meteorological Society, 136: 1530–1542. doi: 10.1002/qj.572	F

`l_urban_empirical`	logical	Switch to use empirical relationships for urban geometry, based on total urban fraction. Dimensions calculated are W/R, H/W & H (see Table 38) URBAN-2T uses W/R only. Used in calculation of the canyon and roof fractions and also to distribute anthropogenic heat between roof and canyon if `l_anthrop_heat_src = TRUE` TRUE: Use empirical relationships for urban geometry. FALSE: Appropriate data needs to be supplied instead NB: These are only valid for high resolutions (~1 km) References: Bohnenstengel SI, Evans S, Clark P, Belcher SE (2010). *Simulations of the London urban heat island*, Quarterly Journal of the Royal Meteorological Society (submitted)	T
The following are the parameterisation switches for the configuration of MORUSES. Where appropriate Table 34 gives the `urban_roof` and `urban_canyon` parameters that are required to be set in `nveg_params.nml`.			
`l_moruses_macdonald`	logical	MORUSES switch for using MacDonald et al. (1998) to calculate effective roughness length of urban areas and displacement height from urban geometry (H, H/W and W/R, see Table 38). TRUE: Use MacDonald et al. (1998) formulations. FALSE: Appropriate data needs to be supplied instead. NB: If `l_urban_empirical` = TRUE then `l_moruses_macdonald` = TRUE, which the code enforces this. References: Macdonald RW, Griffiths RF, Hall D. 1998. *An improved method for the estimation of surface roughness of obstacle arrays*. Atmos. Env. 32: 1857–1864	T

`l_moruses_albedo`	logical	MORUSES switch for effective canyon albedo parameterisation. The roof albedo is given by `nveg_params.nml`. TRUE: Use MORUSES parameterisation. Requires that `l_cosz` = TRUE, which the code automatically enables. FALSE: See Table 34.	T
`l_moruses_emissivity`	logical	MORUSES switch for effective canyon emissivity parameterisation. The roof emissivity is given by `nveg_params.nml`. TRUE: Use MORUSES parameterisation. FALSE: See Table 34.	F
`l_moruses_rough`	logical	MORUSES switch for effective roughness length for heat parameterisation. TRUE: Use MORUSES parameterisation. FALSE: See Table 34.	T
`l_moruses_storage`	logical	MORUSES switch for thermal inertia and coupling with underlying soil parameterisation. TRUE: Use MORUSES parameterisation. FALSE: See Table 34.	T
`l_moruses_storage_thin`	logical	MORUSES switch to use a thin roof to simulate the effects of insulation. Only used if `l_moruses_storage` = TRUE. TRUE: Use thin, insulated roof. FALSE: Use damping depth based on diffusivity of roofing materials.	T

5.12.2. `URBAN2T_PARAM` namelist members

Table 36 Members of the URBAN_2TPARAM namelist

Name	Type	Notes	Default value
`anthrop_heat_scale`	real	Distribution scaling factor, which allows the anthropogenic heat flux to be spread between the `urban_canyon` and `urban_roof` tiles such that: H_roof = anthrop_heat_scale × H_canyon H_canyon × (W/R) + H_roof × (1.0 – W/R) = anthrop_heat Has a value 0.0 - 1.0 where the extremes correspond to: 0.0 = all released within the canyon 1.0 = evenly spread between canyon and roof Only used if `l_anthrop_heat_src` = TRUE	1.0

5.12.3. `URBAN_PROPERTIES` namelist members

Table 37 Members of the URBAN_PROPERTIES namelist

Name	Type	Notes	Default value
`file`	character	The file to read urban properties from. If `use_file` (see below) is FALSE for every variable, this will not be used. This file name can use variable name templating (see Section 4.5.2).	None
`nvars`	integer >= 0	The number of urban property variables that will be provided. The required variables depend on whether MORUSES is used or not. If MORUSES is on, all variables must be given. However, depending on the configuration of MORUSES, not all given variables will be used - see Table 38. Those that will not be used could be set to constant values to avoid setting them from file. If MORUSES is not on, only wrr is required.	0
`var`	character(nvars)	List of urban property variable names as recognised by JULES (see Table 38). Names are case sensitive. *For ASCII files, variable names must be in the order they appear in the file.*	None

`use_file`	logical(nvars)	For each JULES variable specified in `var`, this indicates if it should be read from the specified file or whether a constant value is to be used. TRUE: the variable will be read from the file. FALSE: the variable will be set to a constant value everywhere using `const_val` below.	T
`var_name`	character(nvars)	For each JULES variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data. This is not used for variables where `use_file` = FALSE, but must still be given. *For ASCII files, this is not used - only the order in the file matters, as described above.*	None
`tpl_name`	character(nvars)	For each JULES variable specified in `var`, this is the string to substitute into the file name in place of the variable name substitution string. If the file name does not use variable name templating, this is not used.	None
`const_val`	real(nvars)	For each JULES variable specified in `var` where `use_file` = FALSE, this is a constant value that the variable will be set to at every point. This is not used for variables where `use_file` = TRUE, but must still be given.	None

All of the urban property variables listed below are expected to have no additional dimension.

Table 38 Urban property variables for description of urban geometry & building material variables

Variable name	**Description**[3]	**Notes on when data is not used. If not used / updated with calculated values then the variable could be set to `const_val` instead.**
`wrr`	Repeating width ratio (or canyon fraction, W/R)	If `l_urban_empirical` = TRUE then this is updated with calculated values.
The following refer to MORUSES only		
`hwr`	Height-to-width ratio (H/W)	See for `wrr` above.
`hgt`	Building height (H)	See for `wrr` above.
`ztm`	Effective roughness length of urban areas	If `l_moruses_macdonald` = TRUE (or `l_urban_empirical` = TRUE) then this is updated with calculated values.
`disp`	Displacement height	See for `ztm` above.
`albwl`	Wall albedo	Data only used if `l_moruses_albedo` = TRUE.
`albrd`	Road albedo	See for `albwl` above.

[3] For more information on the urban geometry used please see the JULES technical documentation

`emisw`	Wall emissivity	Data only used if `l_moruses_emissivity` = TRUE.
`emisr`	Road emissivity	See for `emisw` above.

5.13. `imogen.nml`: Settings for the IMOGEN impacts model

This file contains two namelists called `IMOGEN_RUN_LIST` and `IMOGEN_ANLG_VALS_LIST`. Values from this section are only used if IMOGEN is enabled (`l_imogen` = TRUE - see Section 5.2).

A full IMOGEN experiment consists of three JULES runs, called spin_eq, spin_dyn and tran. Each run is started from the final dump of the previous run. The first two runs behave like an extended spin-up for the full transient run, and as such runs with IMOGEN must have no spin-up (i.e. `max_spinup_cycles` = 0 in `timesteps.nml` - see Section 5.4). Since IMOGEN calculates the forcing for an entire year at once, an IMOGEN run must have a start time of 00:00:00 on the 1st of January for some year.

IMOGEN is currently restricted to run only on the HadCM3LC grid, i.e. there are 96 x 56 grid cells where each cell has size 3.75 degrees longitude by 2.5 degrees latitude with no Antartica. This means that:

- `nx` = 96 and `ny` = 56 in the `JULES_INPUT_GRID` namelist.
- The file `examples/imogen/data/jules/grid_info.nc` should be used to set up the correct latitude, longitude and land fraction, as in the given example.

IMOGEN also uses its own I/O, so it expects IMOGEN specific files in a different format to JULES - this may change in the future. An example of setting up IMOGEN is provided in `examples/imogen` - this should be used as a guide if creating new files for IMOGEN.

5.13.1. `IMOGEN_RUN_LIST` namelist members

Table 39 Members of the IMOGEN_RUN_LIST namelist

Name	Type	Notes	Default value
`co2_init_ppmv`	real	Initial CO2 concentration (ppmv)	286.085
`file_scen_emits`	character	If used, file containing CO2 emissions. This file is expected to be in a specific format – see the IMOGEN example	None
`file_non_co2_vals`	character	If used, file containing non-CO2 values. This file is expected to be in a specific format – see the IMOGEN example	None
`file_scen_co2_ppmv`	character	If used, file containing CO2 values. This file is expected to be in a specific format – see the IMOGEN example	None
`anlg`	logical	If TRUE, then use the GCM analogue model	T
`anom`	logical	If TRUE, then incorporate anomalies	T
`c_emissions`	logical	If TRUE, means CO2 concentration is calculated	T
`include_co2`	logical	If TRUE, include adjustments to CO2 values	T
`include_non_co2`	logical	If TRUE, include adjustments to non-CO2 values	T
`land_feed`	logical	If TRUE, include land feedbacks on atmospheric CO2	F

`ocean_feed`	logical	If TRUE, include ocean feedbacks on atmospheric CO2	F
`nyr_emiss`	integer	Number of years of emission data in file	241
`file_points_order`	character	File containing the mapping of IMOGEN global grid points onto IMOGEN land points (different from the JULES land points).	None
`initialise_from_dump`	logical	Indicates how the IMOGEN prognostic variables will be initialised. TRUE: Use a dump file (specified in dump_file below) from a previous run with IMOGEN to initialise the IMOGEN prognostics. FALSE: IMOGEN will handle the initialisation of IMOGEN prognostics.	F
`dump_file`	character	Only used if `initialise_from_dump` is TRUE. The name of the dump file to initialise from.	None

5.13.2. `IMOGEN_ANLG_VALS_LIST` namelist members

Table 40 Members of the IMOGEN_ANLG_VALS_LIST namelist

Name	Type	Notes	Default value
`q2co2`	real	Radiative forcing due to doubling CO2 (W/m2)	3.74
`f_ocean`	real	Fractional coverage of the ocean	0.711
`kappa_o`	real	Ocean eddy diffusivity (W/m/K)	383.8
`lambda_l`	real	Inverse of climate sensitivity over land (W/m2/K)	0.52
`lambda_o`	real	Inverse of climate sensitivity over ocean (W/m2/K)	1.75
`mu`	real	Ratio of land to ocean temperature anomalies	1.87
`t_ocean_init`	real	Initial ocean temperature (K)	289.28
`nyr_non_co2`	integer	Number of years for which non-co2 forcing is prescribed	21
`dir_patt`	character	Directory containing the patterns. Files in this directory are expected to be in a specific format – see the IMOGEN example	None
`dir_clim`	character	Directory containing initialising climatology. Files in this directory are expected to be in a specific format – see the IMOGEN example	None
`dir_anom`	character	Directory containing prescribed anomalies. Files in this directory are expected to be in a specific format – see the IMOGEN example	None

`file_non_co2`	logical	If true, then non-CO2 radiative forcings are contained within a file.	F

5.14. drive.nml: Meteorological driving data

This file contains a single namelist called JULES_DRIVE. that indicates how meteorological driving data is input.

5.14.1. JULES_DRIVE namelist members

If IMOGEN is enabled (l_imogen = TRUE), then meteorological forcing is provided by IMOGEN. In this case, only z1_tq_in and z1_uv_in are used from this namelist.

Table 41 Members of the JULES_DRIVE namelist

Name	Type	Notes	Default value
z1_uv_in	real > 0.0	The height (m) at which the wind data are valid. This height is relative to the zero-plane *not* the ground.	10.0
z1_tq_in	real > 0.0	The height (m) at which the temperature and humidity data are valid. This height is relative to the zero-plane *not* the ground.	10.0
t_for_snow	real	If total precipitation is given as a forcing variable, then t_for_snow is the near-surface air temperature (K) at or below which the precipitation is assumed to be snowfall. At higher temperatures, all the precipitation is assumed to be liquid.	274.0
t_for_con_rain	real	If total preciption or total rainfall are given, then t_for_con_rain is the near-surface air temperature (K) at or above which rainfall is assumed to be convective in origin. At lower temperatures, all the rainfall is assumed to be large-scale in origin. Also see confrac in Section 5.6.2). t_for_con_rain is not used if l_point_data is TRUE, since then there is no convective precipitation. All snow is assumed to be large-scale in origin.	373.15
diff_frac_const	real	A constant value used to calculate diffuse radiation from the total downward shortwave radiation. Only used if diffuse radiation is not given as a forcing variable (see Table 42).	0.0
Members used to specify the start, end and period of the data			

`data_start` `data_end`	character	The times of the start of the first timestep of data and the end of the last timestep of data. Each run of JULES (configured in `timesteps.nml` - see Section 5.4) can use part or all of the specified data. However, there must be data for all times between run start and run end (determined by `main_run_start`, `main_run_end`, `spinup_start` and `spinup_end`). The times must be given in the format **yyyy-mm-dd hh:mm:ss**	None
`data_period`	integer -1, -2 or > 0	The period (in seconds) of the data. Special cases: -1: Monthly data -2: Yearly data	None
Members used to specify the files containing the data			
`read_list`	logical	Switch controlling how data file names are determined for a given time. TRUE: Use a list of data file names with times of first data. FALSE: Use a single data file for all times or a template describing the names of the data files.	F
`nfiles`	integer >= 0	Only used if `read_list` = TRUE. The number of data files to read name and time of first data for.	0
`file`	character	If `read_list` = TRUE, this is the file to read the list of data file names and times from. Each line should be of the form: `'/data/file', 'yyyy-mm-dd hh:mm:ss'` The data file names may contain variable name templating, with the proviso that either no file names use variable name templating or all file names do. The files must appear in chronological order. If `read_list` = FALSE, this is either the single data file (if no templating is used) or a template for data file names. Both time and variable name templating may be used (see Section 4.5).	None
Members used to specify the provided variables			
`nvars`	integer >= 0	The number of forcing variables that will be provided. See Table 42 for the available forcing variables and their possible configurations.	0

`var`	character(nvars)	List of forcing variable names as recognised by JULES (see Table 42). Names are case sensitive. *For ASCII files, variable names must be in the order they appear in the file.*	None
`var_name`	character(nvars)	For each JULES variable specified in `var`, this is the name of the variable in the file(s) containing the data. *For ASCII files, this is not used - only the order in the file matters, as described above.*	None
`tpl_name`	character(nvars)	For each JULES variable specified in `var`, this is the string to substitute into the file name(s) in place of the variable name substitution string. If the file name(s) do not use variable name templating, this is not used.	None
`interp`	character(nvars)	For each JULES variable specified in `var`, this indicates how the variable is to be interpolated in time (see Section 4.6).	None

All of the available forcing variables listed below are expected to have no additional dimension.

Table 42 List of JULES forcing variables

Name	Description
Always required	
`pstar`	Air pressure (Pa)
`q`	Specific humidity (kg kg^{-1})
`t`	Air temperature (K)
Radiaton variables The radiation forcing variables can be given in one of three ways: 1. **`rad_net` and `sw_down`:** Downward shortwave and net (all wavelengths) downward radiation are input. The modelled albedo and surface temperature are used to calculate the downward longwave flux. 2. **`lw_net` and `sw_net`:** Net downward fluxes of short- and longwave radiation are input. The modelled albedo and surface temperature are used to calculate the downward fluxes of shortwave and longwave radiation. 3. **`sw_down` and `lw_down`:** Downward fluxes of short- and longwave radiation are input. Normally this is the preferred option. If `rad_net` is present in the given list of variables, the first method is used. If `rad_net` is *not* present but `lw_net` is, then the second method is used. The third method is used in all other cases. `diff_rad` can be used with any of the three methods. If it is given, diffuse radiation is input from file. If it is not given, `diff_frac_const` is used instead to partition the downward shortwave radiation into diffuse and direct.	
`rad_net`	Net (all wavelength) downward radiation (W m^{-2}) .
`lw_net`	Net downward longwave radiation (W m^{-2}).

`sw_net`	Net downward shortwave radiation (W m^{-2}).
`lw_down`	Downward longwave radiation (W m^{-2}).
`sw_down`	Downward shortwave radiation (W m^{-2}).
`diff_rad`	Diffuse radiation (W m^{-2}).

Precipitation variables

The precipitation variables can be specified in one of four ways:

1. **`precip`:** A single precipitation field is input. This represents the total precipitation (rainfall and snowfall). The total is partitioned between snowfall and rainfall using `t_for_snow` (see above), and rainfall is then further partitioned into large-scale and convective components using `t_for_con_rain` (see above). Convective snowfall is assumed to be zero.
2. **`tot_rain` and `tot_snow`:** Two precipitation fields are input, namely total rainfall and total snowfall. The rainfall is partitioned between large-scale and convective, using `t_for_con_rain` (see above). Convective snowfall is assumed to be zero.
3. **`ls_rain`, `con_rain` and `tot_snow`:** Three precipitation fields are input, namely large-scale rainfall, convective rainfall and total snowfall. This cannot be used with `l_point_data` = TRUE. Convective snowfall is assumed to be zero.
4. **`ls_rain`, `con_rain`, `ls_snow` and `con_snow`:** Four precipitation fields are input, namely large-scale rainfall, convective rainfall, large-scale snowfall and convective snowfall. This cannot be used with `l_point_data` = TRUE. Note that this is the only option that considers convective snowfall.

If `precip` is given, the first method is used. If `precip` is *not* given but `tot_rain` is, the second method is used. If *neither* `precip` *nor* `tot_rain` are given but `tot_snow` is, the third method is used. The fourth method is used in all other cases.

The concept of convective and large-scale (or dynamical) components of precipitation comes from atmospheric models, in which the precipitation from small-scale (convective) and large-scale motions is often calculated separately. If JULES is to be driven by the output from such a model, the driving data might include these components.

`precip`	Precipitation rate (kg m^{-2} s^{-1}).
`tot_rain`	Rainfall rate (kg m^{-2} s^{-1}).
`tot_snow`	Snowfall rate (kg m^{-2} s-1).
`ls_rain`	Large-scale rainfall rate (kg m^{-2} s^{-1}).
`con_rain`	Convective rainfall rate (kg m^{-2} s^{-1}).
`ls_snow`	Large-scale snowfall rate (kg m^{-2} s^{-1}).
`con_snow`	Convective snowfall rate (kg m^{-2} s^{-1}).

Wind variables

The wind variables can be given in one of two ways:

1. **`wind`:** The wind speed is input.
2. **`u` and `v`:** The two components of the horizontal wind (e.g. the southerly and westerly components) are input.

If `wind` is given, then the first method is used. The second method is used in all other cases.

`wind`	Total wind speed (m s^{-1}).
`u`	Zonal component of the wind (m s^{-1}).
`v`	Meridional component of the wind (m s^{-1}).

5.14.2. Examples of specifying driving data

Single point ASCII driving data for one year

```
&JULES_DRIVE
  diff_frac_const = 0.1,

  data_start  = '1997-01-01 00:00:00',
  data_end    = '1998-01-01 00:00:00',
  data_period = 1800,

  file = "met_data.dat",

  nvars = 8,
  var    = 'sw_down' 'lw_down' 'tot_rain' 'tot_snow'  't' 'wind' 'pstar'  'q',
  interp =      'nf'      'nf'       'nf'       'nf' 'nf'   'nf'    'nf' 'nf'
/
```

`data_start`, `data_end` and `data_period` specify that the driving dataset provides one year (1997) of half-hourly data.

`read_list` is not given, so takes its default value of FALSE. This means that file is used as either the single data file or a file name template. In this case there is no templating, so JULES treats the given file as the single data file for all data times.

Neither `rad_net` nor `lw_net` are given, so radiation scheme 3 (above) is used.

`precip` is not given but `tot_rain` is, so precipitation scheme 2 (above) is used.

`wind` is given, so wind speed is used (scheme 1 above).

`diff_rad` is not given, so the diffuse radiation is calculated as 0.1 (the value of `diff_frac_const`) times the total shortwave radiation.

An example of the driving data is given in Figure 3.

Figure 3 Lines of an example meteorological driving data in ASCII format

```
# solar   long  rain  snow    temp   wind     press      humid
    3.3  187.8   0.0   0.0  259.10  3.610  102400.5  1.351E-03
   89.5  185.8   0.0   0.0  259.45  3.140  102401.9  1.357E-03
  142.3  186.4   0.0   0.0  259.85  2.890  102401.0  1.369E-03
----- data for later times ----
```

Driving data from NetCDF files with one variable per file

```
&JULES_DRIVE
  data_start  = '1982-07-01 03:00:00',
  data_end    = '1996-01-01 00:00:00',
  data_period = 10800,

  read_list = T,
  nfiles    = 162,
  file = "./file_list.txt"

  nvars = 8,
  var      = 'sw_down' 'lw_down' 'tot_rain' 'tot_snow'    't' 'wind' 'pstar'    'q',
  var_name =  'SWdown'  'LWdown'    'Rainf'    'Snowf' 'Tair' 'Wind' 'PSurf' 'Qair',
  tpl_name =  'SWdown'  'LWdown'    'Rainf'    'Snowf' 'Tair' 'Wind' 'PSurf' 'Qair',
  interp   =      'nb'      'nb'       'nb'       'nb'    'i'    'i'     'i'    'i'
/
```

In this example, the driving dataset provides 13.5 years of driving data on a 3 hourly timestep.

`read_list` = TRUE indicates that the names and start times of the data files should be read from `file_list.txt`. The first few lines of this file are shown in Figure 4.

Figure 4 Example list of driving data files using file name templating

```
'met_data/%vv_data/%vv198207.nc', '1982-07-01 03:00:00'
'met_data/%vv_data/%vv198208.nc', '1982-08-01 03:00:00'
'met_data/%vv_data/%vv198209.nc', '1982-09-01 03:00:00'
------ rest of file not shown -----
```

The presence of the variable name templating string in each file name shows that we are using variable name templating (see Section 4.5.2). The dates show that we in fact have monthly files, but we cannot use time templating for these files because the start time of 03H does not conform to the requirements. Furthermore, files for each variable are stored in separate directories. The values from `tpl_name` will be substituted into the file name templates in place of the substitution string. For example, pressure is held in files that look like `‘met_data/PSurf_data/PSurf198207.nc’`, and temperature in files like `‘met_data/Tair_data/Tair198207.nc’`.

The driving variable setup is as the previous example, except that `diff_frac_const` takes its default value of 0.0.

5.15. `prescribed_data.nml`: Other time-varying input data

This file contains a variable number of namelists that are used to prescribe time-varying input data that is not meteorological forcing. The namelist `JULES_PRESCRIBED` should occur only once at the top of the file. The value of `n_datasets` in `JULES_PRESCRIBED` then determines how many times the namelist `JULES_PRESCRIBED_DATASET` should occur.

5.15.1. `JULES_PRESCRIBED` namelist members

Table 43 Members of the JULES_PRESCRIBED namelist

Name	Type	Notes	Default value
`n_datasets`	integer >= 0	The number of datasets that will be specified using instances of the `JULES_PRESCRIBED_DATASET` namelist.	0

5.15.2. `JULES_PRESCRIBED_DATASET` namelist members

This namelist should occur `n_datasets` times. Each occurrence of this namelist contains information about a single dataset (i.e. set of related files).

Table 44 Members of the JULES_PRESCRIBED_DATASET namelist

Name	Type	Notes	Default value
Members used to specify the start, end and period of the data			
`data_start` `data_end`	character	The time of the start of the first timestep of data and the end of the last timestep of data. Each run of JULES (configured in `timesteps.nml` - see Section 5.4) can use part or all of the specified data. However, there must be data for all times between run start and run end (determined by `main_run_start`, `main_run_end`, `spinup_start` and `spinup_end`). The only time that the specified data times can cover less than the entire run is in the case of a climatology (see below), in which case exactly a year of data must be provided, and will be cycled. The times must be given in the format **yyyy-mm-dd hh:mm:ss**	None
`data_period`	integer -1, -2 or > 0	The period (in seconds) of the data. Special cases: -1: Monthly data -2: Yearly data	None

`is_climatology`	logical	Indicates whether the data is to be used as a climatology (use the same data for every year). TRUE: Interpret the data as a climatology. Exactly one year of data must be specified. FALSE: Do not interpret the data as a climatology.	F
Members used to specify the files containing the data			
`read_list`	logical	Switch controlling how data file names are determined for a given time. TRUE: Use a list of data file names with times of first data. FALSE: Use a single data file for all times or a template describing the names of the data files.	F
`nfiles`	integer >= 0	Only used if `read_list` = TRUE. The number of data files to read name and time of first data for.	0
`file`	character	If `read_list` = TRUE, this is the file to read the list of data file names and times from. Each line should be of the form: `'/data/file', 'yyyy-mm-dd hh:mm:ss'` The data file names may contain variable name templating, with the proviso that either no file names use variable name templating or all file names do. The files must appear in chronological order. If `read_list` = FALSE, this is either the single data file (if no templating is used) or a template for data file names. Both time and variable name templating may be used (see Section 4.5).	None
Members used to specify the provided variables			
`nvars`	integer >= 0	The number of variables that the dataset will provide. See Table 45 for the supported variables.	0
`var`	character(nvars)	List of variable names as recognised by JULES (see Table 45 for the supported variables). Names are case sensitive. *For ASCII files, variable names must be in the order they appear in the file.*	None
`var_name`	character(nvars)	For each JULES variable specified in `var`, this is the name of the variable in the file(s) containing the data. *For ASCII files, this is not used - only the order in the file matters, as described above.*	None

`tpl_name`	character(nvars)	For each JULES variable specified in `var`, this is the string to substitute into the file name(s) in place of the variable name substitution string. If the file name(s) do not use variable name templating, this is not used.	None
`interp`	character(nvars)	For each JULES variable specified in `var`, this indicates how the variable is to be interpolated in time (see Section 4.6)	None

In theory, any variable with an entry in the subroutine `populate_var` in `model_interface_mod` (see Section 7.1) can be updated via this mechanism, and the use of any of these variables is not explicitly prevented. However, it is up to the user to assess whether using this mechanism to update any particular variable is desirable. The use of the following variables *is* explicitly supported:

Table 45 Supported variables for prescribed datasets

Name	Description	Size of additional dimension
`ozone`	Surface ozone concentration (ppb) This is required if `l_o3_damage` is TRUE.	None
`canht`	PFT canopy height (m)	npft
`lai`	PFT leaf area index	npft

5.16. `initial_conditions.nml`: Specification of the initial state

This file contains a single namelist called `JULES_INITIAL` that used to set up the initial state of prognostic variables.

5.16.1. `JULES_INITIAL` namelist members

The values of all prognostic variables must be set at the start of a run. This initial state, or initial condition, can be read from a "dump" from an earlier run of the model, or may be read from a different file. Another option is to prescribe a simple or idealised initial state by giving constant values for the prognostic variables directly in the namelist. It is also possible to set some fields using values from a file (e.g. a dump) but to set others to constants given in the namelist.

Table 46 Members of the JULES_INITIAL namelist

Name	Type	Notes	Default value
`dump_file`	logical	Indicates whether the given file (see below) is a dump from a previous run of the model. TRUE: The file is a JULES dump file. FALSE: The file is not a JULES dump file.	F
`total_snow`	logical	Switch controlling simplified initialisation of snow variables. `total_snow` must be TRUE unless initialising from a dump file to ensure that the initialisation of snow variables is consistent. TRUE: Only the total mass of snow on each tile (see `snow_tile` in Table 47) is required to be input, and all related variables will be calculated from this or simple assumptions made. All the snow is assumed to be on the ground (not in the canopy). If `dump_file` = FALSE, `total_snow` must be TRUE. FALSE: All snow variables required for the current configuration must be input separately (see Table 47). This is only allowed if `dump_file` = TRUE.	T
Members used to set up spacially varying properties			
`file`	character	The file to read initial conditions from. This can be a JULES dump file or a file conforming to the JULES requirements (see Section 4). If `use_file` (see below) is FALSE for every variable, this will not be used. This file name can use variable name templating (see Section 4.5.2).	None

`nvars`	integer >= 0	The number of initital condition variables that will be provided. See Table 47 for those required for a particular configuration. If `dump_file` = TRUE and `nvars` = 0, then the model will attempt to initialise all required variables from the given dump file.	0
`var`	character(nvars)	List of initial condition variable names as recognised by JULES (see Table 47). Names are case sensitive. *For ASCII files, variable names must be in the order they appear in the file.*	None
`use_file`	logical(nvars)	For each JULES variable specified in `var`, this indicates if it should be read from the specified file or whether a constant value is to be used. TRUE: the variable will be read from the file. FALSE: the variable will be set to a constant value everywhere using `const_val` below.	T
`var_name`	character(nvars)	For each JULES variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data. This is not used for variables where `use_file` = FALSE, but must still be given. *For ASCII files, this is not used - only the order in the file matters, as described above.*	None
`tpl_name`	character(nvars)	For each JULES variable specified in `var`, this is the string to substitute into the file name in place of the variable name substitution string. If the file name does not use variable name templating, this is not used.	None
`const_val`	real(nvars)	For each JULES variable specified in `var` where `use_file` = FALSE, this is a constant value that the variable will be set to at every point. This is not used for variables where `use_file` = TRUE, but must still be given.	None

All input to the model must be on the same grid (see Sections 4 and 5.5), and initial conditions are no different. Even when the variable is only required for land points, values must be provided for the full input grid. The required variables for a particular configuration, along with the expected size of their additional dimension, are given in the table below. The dimension sizes are given in terms of the following variables:

- `sm_levels` - the number of soil layers (see Section 5.3).
- `npft` – the number of plant functional types (see Section 5.3).
- `ntype` - the number of surface types (`npft + nnvg` - see Section 5.3).
- `ntiles` - the number of tiles at each gridbox (1 if `l_aggregate` = TRUE, `ntype` otherwise).
- `sc_pools` - the number of soil carbon pools (1 if `l_triffid` = FALSE, 4 if `l_triffid` = TRUE).

Table 47 List of initial condition variables

Name	Description	Size of additional dimension
Always required		
`canopy`	Amount of intercepted water that is held on each tile (kg m^{-2}).	`ntiles`
`cs`	Soil carbon (kg m^{-2}). If TRIFFID is not being used (`l_triffid` = FALSE), this is the total soil carbon. If TRIFFID is being used, this is the carbon in each of the 4 pools of the RothC model.	`sc_pools`
`gs`	Stomatal conductance for water vapour (m s^{-1}). This is used to start the iterative calculation of `gs` for the first timestep only.	None
`snow_tile`	If `can_model` $\neq$ 4, this is the total snow on the tile (since there is a single store which doesn't distinguish between snow on canopy and under canopy). If `can_model` = 4 (and then only at tiles where `cansnowpft` = TRUE), `snow_tile` is interpreted as the snow on the canopy, except as overridden by `total_snow` = TRUE. If `total_snow` = TRUE, `snow_tile` is used to hold the total snow on the tile (and is subsequently put onto the ground at tiles that distinguish between ground and canopy stores). Further details of snow initialisation are given below.	`ntiles`
`sthuf`	Soil wetness for each soil layer. This is the mass of soil water (liquid and frozen), expressed as a fraction of the water content at saturation.	`sm_levels`
`t_soil`	Temperature of each soil layer (K).	`sm_levels`
`tstar_tile`	Temperature of each tile (K). This is the surface or skin temperature.	`ntiles`
Required if `l_phenol` = TRUE		
`lai`	Leaf area index of each PFT.	`npft`
Required if `l_triffid` = TRUE		
`canht`	Height (m) of each PFT.	`npft`
Required if `l_veg_compete` = TRUE		
`frac`	The fraction of land area of each gridbox that is covered by each surface type.	`ntype`
Required if `l_top` = TRUE		
`sthzw`	Soil wetness in the deep ("water table") layer beneath the	None

	standard soil column This is the mass of soil water (liquid and frozen), expressed as a fraction of the water content at saturation.	
`zw`	Depth from the surface to the water table (m).	None
Required if `l_spec_albedo` = TRUE		
`rgrain`	Snow surface grain size (μm) on each tile.	`ntiles`
All variables from here onwards are only required if `total_snow` = FALSE, but different configurations require different variables. **These variables can only be set from a dump file or to constant values.**		
`rho_snow`	Bulk density of lying snow (kg m^{-3}). If `total_snow` = TRUE then this is set as follows: • If `nsmax` = 0, it is set to `rho_snow_const`. • If `nsmax` > 0 and there is an existing snow pack, it is set to `rho_snow_const`. • If `nsmax` > 0 and there is no snow pack, it is set to `rho_snow_fresh`.	N/A - variable is either derived from snow_tile, set from dump or set to a constant value
`snow_depth`	Depth of snow (kg m). If `total_snow` = TRUE, this is calculated from mass and density of snow.	N/A - variable is either derived from snow_tile, set from dump or set to a constant value
Required if `total_snow` = FALSE and `can_model` = 4		
`snow_grnd`	Amount of snow on the ground, beneath the canopy (kg m^{-2}), on each tile. If `total_snow` = TRUE this is set to `snow_tile` at tiles where `can_model` = 4 is active, and to zero at all other tiles.	N/A - variable is either derived from snow_tile, set from dump or set to a constant value
Required if `total_snow` = FALSE and `nsmax` > 0		
`nsnow`	The number of snow layers on each tile. If `total_snow` = TRUE this is calculated from the snow depth.	N/A - variable is either derived from snow_tile, set from dump or set to a constant value
`snow_ds`	Depth of snow in each layer (kg m). If `total_snow` = TRUE this is calculated from the snow depth and the number of snow layers.	N/A - variable is either derived from snow_tile, set from dump or set to a constant value
`snow_ice`	Mass of frozen water in each snow layer (kg m^{-2}). If `total_snow` = TRUE all snow is assumed to be ice.	N/A - variable is either derived from snow_tile, set from dump or set to a constant value
`snow_liq`	Mass of liquid water in each snow layer (kg m^{-2}). If `total_snow` = TRUE this is set to zero.	N/A - variable is either derived from snow_tile, set from dump or set to a constant value

tsnow	Temperature of each snow layer (K). If total_snow = TRUE this is set to the temperature of the top soil layer.	N/A - variable is either derived from snow_tile, set from dump or set to a constant value
Required if total_snow = FALSE, nsmax > 0 and l_spec_albedo = TRUE		
rgrainl	Snow grain size (µm) on each tile in each snow layer. If total_snow = TRUE this is set to rgrain.	N/A - variable is either derived from snow_tile, set from dump or set to a constant value

5.16.1.1. Examples of specification of initial state

Specification of initial state at a single point

This assumes that l_phenol = FALSE, l_triffid = FALSE and nsmax = 0.

```
&JULES_INITIAL
  file = "initial_conditions.dat",

  nvars = 8,
  var       = 'canopy'  'tstar_tile'   'cs'  'gs'  'rgrain'  'snow_tile'  'sthuf'  't_soil',
  use_file  =       F              F      F     F         F            F        T         T ,
  const_val =     0.0         276.78   12.1   0.0      50.0          0.0
/
```

This shows how a mixture of constant values and initial state from a file can be used. In this case, the first 6 variables will be set to constant values everywhere (use_file = FALSE) with the last 2 read from the specified file (use_file = TRUE).

file specifies an ASCII file to read the variables for which use_file = TRUE from.

Since the variables are arranged such that all those with use_file = FALSE first, we need only supply constant values for those variables that require them.

The contents of the specified file are shown in Figure 5.

Figure 5 Contents of initial_conditions.dat

```
# sthuf(1:4)                      t_soil(1:4)
  0.749  0.743  0.754  0.759      276.78  277.46  278.99  282.48
```

The data for each soil layer is given in consecutive columns. A comment line is used to indicate which columns comprise which variable (see Section 4 for more details).

Specifying initial state for gridded data using NetCDF files is similar, except that variable names (var_name in Table 46) are also required for each variable.

Specification of initial state from an existing dump file

In this example, we use an existing dump file (from a previous run) to set the initial values of all required variables.

```
&JULES_INITIAL
  total_snow = F,
  dump_file  = T,
  file       = "jules_dump.nc"
/
```

`total_snow` = FALSE indicates that we want to initialise all the snow variables directly from the dump.

`dump_file` = TRUE indicates that the given file should be interpreted as a JULES dump file.

`file` specifies the dump file to read (in this case a NetCDF dump file).

Since it is not specified, `nvar`s takes its default value of 0, which indicates that JULES should attempt to read all required variables from the given dump file.

5.17. `output.nml`: Specification of output from the model

This file contains a variable number of namelists that are used to specify the output required by the user. The namelist `JULES_OUTPUT` should occur only once at the top of the file. The value of `nprofiles` in `JULES_OUTPUT` then determines how many times the namelist `JULES_OUTPUT_PROFILE` should occur.

JULES separates output into one or more output 'profiles' or streams. Within each profile, all variables selected for output are written to the same file, with the same frequency, although the time-processing can differ between variables (e.g. instantaneous values and time-averages can appear in the same profile). Each profile can be considered as a separate data stream. By using more than one profile the user can, for example:

- Output one set of variables to one file, and other variables to another file.
- Write instantaneous values to one file, and time-averaged values to another.
- Write low-frequency output throughout the run to one file, and high-frequency output from a smaller part of the run (e.g. a "Special Observation period") to another file.

All output is provided on the model grid only (see Section 5.5). Each output file contains the latitude and longitude of each point to allow the points to be located in a grid if desired (e.g. for visualisation). Each output file contains a time indexing variable to locate the values in time.

JULES also writes dump files (a snapshot of the current model state) at the following times:

- After initialisation is complete, immediately before the start of the run (initial state).
- At the end of each cycle of spin-up.
- At the end of the main run.
- At the end of each calendar year.

Each dump is marked with the model date and time that it was produced.

All output files will be NetCDF if JULES is compiled with the 'proper' NetCDF libraries (see Section3.2). Otherwise all output will be in columnar ASCII files.

5.17.1. `JULES_OUTPUT` namelist members

Table 48 Members of the JULES_OUTPUT namelist

Name	Type	Notes	Default value
`output_dir`	character	The directory used for output files. This can be an absolute or relative path.	None
`run_id`	character	A name or identifier for the run. This is used to name output files and model dumps.	None
`nprofiles`	integer >= 0	The number of output profiles that will be specified using instances of the `JULES_OUTPUT_PROFILE` namelist.	0

5.17.2. `JULES_OUTPUT_PROFILE` namelist members

This namelist should occur `nprofiles` times. Each occurrence of this namelist contains information about a single output profile, as described above.

Table 49 Members of the JULES_OUTPUT_PROFILE namelist

Name	Type	Notes	Default value
`profile_name`	character	The name of the output profile. This is used in file names and should be specified, even if there is only one profile. The names might reflect the variables in the file (e.g. `'soil'`), the data frequency (e.g. `'daily'`), or if several profiles are used they could be given arbitrary names such as `'p1'`, `'p2'`, …, etc.	None
`file_period`	integer 0, -1 or -2	The period for output files, i.e. the time interval within which all output goes to the same file. Note that in all cases, output during spin-up goes into separate files for each spin-up cycle and output during the main run goes into its own file(s). This can be one of three values: 0: All output goes into the same file. -1: Monthly files are produced (i.e. all output for a month goes into the same file). -2: Annual files (calendar years) are produced.	0
Members used to specify the times that the profile will generate output			
`output_spinup`	logical	Determines whether the profile will provide output during model spin-up. TRUE: Provide output during spin-up. Output is provided for the whole of the model spin-up. Output from each spin-up cycle goes into separate files. FALSE: Do not provide any output during spin-up.	F

`output_main_run`	logical	Determines whether the profile will provide output during the main model run (i.e. any part of the run that is not spin-up). TRUE: Provide output during the main model run. Output will be provided for all times between `output_start` and `output_end` below. FALSE: Do not provide any output during the main model run.	F
`output_start` `output_end`	character	The time to start and stop collecting data for output. The times that output is actually produced is determined by `output_period` below. If `output_period` is monthly, then `output_start` must be 00:00:00 on the 1st of some month. If `output_period` is yearly, then `output_start` must be 00:00:00 on the 1st of January for some year. If `output_end` is given such that an output period is not complete when the run reaches `output_end`, output will not be generated for that final period (e.g. if monthly output is being generated but `output_end` is midday on the 31st December, then output will not be generated for December, even though most of December has been run). The times must be given in the format **yyyy-mm-dd hh:mm:ss**	`main_run_start` `main_run_end`

`sample_period`	integer > 1	The sampling period (in seconds) for time-averages and accumulations. This should be a factor of the output period and a multiple of the timestep length. *It is strongly recommended that this be left at the default value (sample every timestep), other than in exceptional cases.* In some cases, sampling every timestep adds a considerable computational burden and acceptable output can be achieved by sampling less frequently. For example, with a large domain, many output diagnostics, and a timestep of 30 minutes, a monthly average would be calculated from several hundred values if every timestep was used. For variables that evolve relatively slowly, an acceptable monthly average might be obtained by sampling only every 12 hours. Note that if fields are not sampled every timestep, the output averages will only be approximations.	`timestep_len`
`output_period`	integer > 1, -1 or -2	The period for output (in seconds). This must be a multiple of the timestep length (except for the special cases <0 given below). Special cases: -1: Monthly period -2: Annual period	`timestep_len`
Members used to specify the variables that will output by this profile			
`nvars`	integer >= 0	The number of variables that the profile should provide output for. The variables available for output are given in Section 1.	0
`var`	character(nvars)	List of variable names to output, as recognised by JULES (see Section 1 for a list of variables available for output). Names are case sensitive.	None
`var_name`	character(nvars)	For each variable specified in `var`, this is the name to give the variable in output files. If the empty string (the default) is given for any variable, then the corresponding value from `var` is used instead.	‘’ (empty string)

`output_type`	character(nvars)	For each variable specified in `var`, this indicates the type of processing required. Recognised values are: • S: Instantaneous or snapshot value. • M: Time mean value. • A: Accumulation over time. For time averaged variables, the period over which each time average is calculated is given by `output_period`. For time-accumulation variables, `output_period` gives the period for output of an updated accumulation (i.e., how often the value if reported). For both time averages and accumulations, the sampling frequency is given by `sample_period`. NB A time-accumulation is initialised at the start of a run and thereafter accumulates until the end of the run. This may mean that accuracy is lost, particularly towards the end of long runs, if small increments are added to an already large sum.	‘S’

5.17.2.1. Example of requesting output

In this example, the user has requested two output profiles. One provides gridbox monthly means for the whole of the main run, the other provides snapshot values of per-tile variables every timestep for a single year. We assume that `timestep_len` = 1800, `main_run_start` = ‘1995-01-01 00:00:00’ and `main_run_end` = ‘2005-01-01 00:00:00’, so that exactly 10 years will be run. There is no spin-up.

```
&JULES_OUTPUT
  run_id = "jules_run001",

  output_dir = "./output",

  nprofiles = 2
/

&JULES_OUTPUT_PROFILE
  profile_name = "month",

  output_main_run = T,

  output_period = -1,

  nvars = 4,
  var         =    "emis_gb"         "ftl_gb"  "snow_mass_gb"  "tstar_gb",
  var_name    = "Emissivity"  "SensibleHeat"      "SnowMass"      "Temp",
  output_type =          'M'             'M'             'M'         'M'
/

&JULES_OUTPUT_PROFILE
  profile_name = "tstep",
  output_main_run = T,
  output_start    = "2000-01-01 00:00:00",
  output_end      = "2001-01-01 00:00:00",

  nvars = 4,
  var         = "emis"  "ftl"  "snow_mass"  "tstar",
  output_type =    'S'    'S'          'S'      'S'
/
```

The JULES_OUTPUT namelist is simple - it gives an id for the run, specifies the directory to put output in and specifies the number of profile definitions that follow.

Each profile is given a unique name. Both profiles want to output part of the main run, so must set output_main_run to TRUE. Since sample_period is not given for either profile, both will use the default (sample every timestep). The same is true for file_period - since it is not given for either profile, it takes its default value and all output for each profile will go into a single file.

The first output profile wants to output monthly averages, so output_period is set to -1 (the special value indicating that calendar months should be used for the output period) and output_type is set to ‘M’ (for mean) for each variable. The user wants this profile to output for the whole of the main run, so does not need to specify output_start or output_end (note that this is only possible because the main run starts at 00:00:00 on the 1st of a month - if this was not the case, the user would have to specify a different time for output_start). The user has also chosen to supply the names that each variable will use in output files using var_name.

The second output profile has specified a section of the main run that it will provide output for using output_start and output_end such that exactly a year of data will be output. Since output_period is not specified, it takes its default value and output will be produced every timestep. The user has not specified the names to use in output files for this profile, so the values from var will be used.

6. JULES example configurations

JULES comes with several example configurations, which can be found in the `examples` directory. These show how to set up the namelist files for various situations. The provided examples are:

1. **`point_loobos`:** A single point simulation forced with weather station data. This run requires a single input file (meteorological data) that is also included as part of the JULES distribution, in the `loobos` directory.
2. **`point_loobos_triffid`:** This is similar to point_loobos, but with the TRIFFID dynamic vegetation model switched on.
3. **`point_VL92_1T`, `point_VL92_2T` and `point_VL92_M`:** These are single point simulations that include the urban land surface types, forced with weather station data (although no data is provided). They are given as examples of setting up the three urban schemes - the original one tile urban scheme, the simple two-tile urban scheme (URBAN-2T) and MORUSES respectively.
4. **`grid_gswp2`:** A gridded domain simulation forced with GSWP2 weather data. This run requires a large amount of input data that is not distributed with JULES, and merely serves as an example of how to set up a run with a gridded domain. A version of the GSWP2 data is distributed with the JULES benchmarking suite, as well as working distributed configurations.

Further example configurations can be seen in the JULES benchmarking suite.

7. Aspects of the code

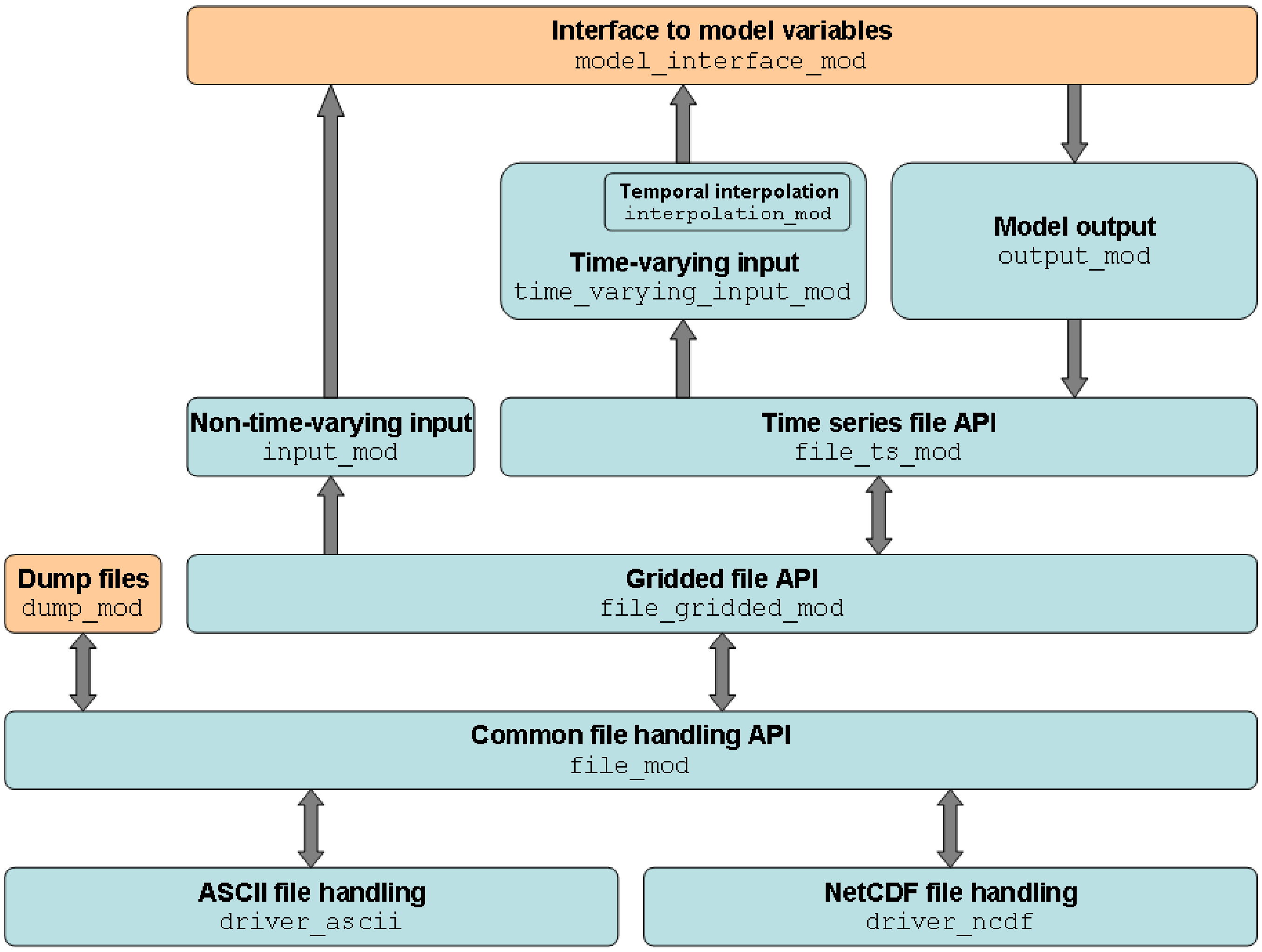
7.1. I/O framework

JULES v3.1 sees a complete rewrite of the I/O code to use a more modular and flexible structure. This section attempts to give a brief description of the low-level I/O framework, and explains how to make some commonly required changes.

7.1.1. Overview

The new I/O code is comprised of several ‘layers’ with clearly defined responsibilities that communicate with each other, as shown in Figure 6 (the relevant Fortran modules for each layer are also given). The blocks in orange are the JULES specific pieces of code - in theory, the rest of the code could be used with other models if different implementations of these modules were provided.

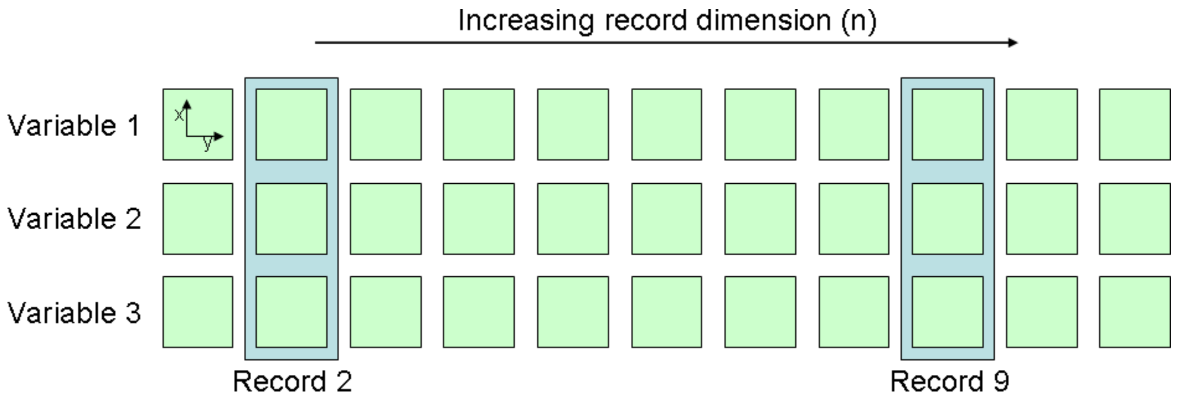
Figure 6 Modular structure of the new I/O code



The core component in the new I/O framework is the common file handling API. This layer provides a common interface for different file formats that is then used by the rest of the code. The drivers for ASCII and NetCDF files implement this interface. The interface is based around the concepts of dimensions and variables, much like NetCDF (except that nothing is inferred from metadata – all information about variables and dimensions must be prescribed), but adds the concept of a ‘record’ to that:

- **Dimension:** A file has one or more dimensions. Each regular dimension has a name and a size. One dimension is special, and is referred to as the record dimension. It has a name but has no defined size. A typical use of the record dimension is to represent time.
- **Variable:** A file has one or more variables. The size of each variable is defined using the dimensions previously defined in the file. Each variable can opt to use the record dimension - if a variable uses the record dimension it must be the last dimension that the variable has.
- **Record:** A record is the collection of all variables at a certain value of the record dimension. Figure 7 gives an example of this - each variable has dimensions x, y and n, where n is the record dimension. Each green box represents the (2D plane of) values of a variable for a certain value of n. A record is then the collection of all variables at a certain value of n. A good analogy is the lines in an ASCII file, where each column represents a variable and each line is a record (in fact, this is a generalisation to multiple dimensions of that exact concept).

Figure 7 Records in a file



Files keep track of the record they are currently pointing at (it is the responsibility of the file-type drivers to do this in the way that best suits the file format they implement). When a file receives a read or write request for a particular variable, the values are read from or written to the current record. The record abstraction also allows two useful operations - seek and advance. When a file receives an instruction to seek to a particular record, it sets its internal pointer so that read/write requests access the given record (a use of this within JULES is looping the input files round spin-up cycles). An advance instruction just moves the internal pointer on to the next record.

The routines in `file_mod` define the interface that each file-type driver must implement, and are responsible for deciding which driver to defer to. Support for a file format is provided by implementing this interface and declaring the implementation in `file_mod`. This is discussed in further detail in Section 7.1.3.

The gridded file API then imposes the concept of reading and writing 3D cubes (x and y dimensions and an optional z dimension) of data on top of the common file handling API. The underlying files may have a 1D or 2D grid (see Section 4), and this layer handles the grid dimensions transparently. It is this layer that handles the extraction of a subgrid from a larger grid (see `file_gridded_read_var` and `file_gridded_write_var`).

The time series file API builds on the gridded file API by explicitly presenting the record dimension as a time dimension. It provides an interface that allows users to treat multiple files (e.g. monthly files, yearly files) as if they were a single file (i.e. seek and advance will automatically open and close files if required).

The input and output layers interact with the model via an interface provided by `model_interface_mod`. `model_interface_mod` allows the input and output layers to read values from and write values to the internal model variables. This is discussed in more detail in Sections 7.1.2 and **Error! Reference source not found.**. The input and output layers use the time series file API to read from and write to file.

This should provide a reasonable introduction to the new I/O code, but looking at the code is the best way to learn about it.

7.1.2. Implementing new variables for input and output

The only I/O code that needs to be modified to add new variables for input and output is in `model_interface_mod` (the routines in `src/io/model_interface`). All interaction between the I/O code and the model happens in this module (apart from reading and writing dump files). Before adding any code to `model_interface_mod`, the variable the user wishes to make available for input and/or output must be accessible to `model_interface_mod` - this is accomplished by placing the variable in a module and importing the module into `model_interface_mod` where required.

There are several places that code must be added in `model_interface_mod` to implement a variable for input or output. Essentially, each routine in `model_interface_mod` uses a `SELECT` statement to choose which variable to operate on - to implement a new variable for input or output, a new `CASE` needs to be added to the appropriate `SELECT` statements. Externally, variables are identified by string identifiers, as seen in this document (e.g. driving variables). Internally, each variable is assigned a unique integer id, which are defined as parameters in `model_interface_mod` - the integer id that the new variable will use should be assigned here, following the system already in place. The routines `get_var_id` and `get_string_identifier` convert back and forth between the string and integer indentifiers for a variable. `get_var_levs_dim` returns the name and size of the z dimension to use for a variable (remember that all input and output uses 3D cubes with x and y dimensions and an optional z dimension). If no z dimension is required, the empty string should be returned as the dimension name and 1 as the dimension size - internally, variables with no z dimension are treated as if they have a z dimension of size 1, but that dimension does not exist in the file(s). A new `CASE` is required in `get_var_id`, `get_string_identifier` and `get_var_levs_dim` for both input and output. To implement a new variable for input, a new `CASE` must also be added to `populate_var` – this is the routine where values from file are used to populate internal model variables. To implement a new variable for output, a new `CASE` statement must also be added in two routines – `get_var_attrs`, where the attributes that will be attached to the variable in output files are defined, and `extract_var`, where values from model variables are extracted on the full model grid ready to output to file. `map_from_land` and `map_to_land` are provided as utilities for use with variables that are defined on land points only.

As always, the best way to go about implementing new variables for input and output is to follow the examples that are already there.

7.1.3. Implementing a new file format

To understand how to implement a new file format, it first helps to understand how the common file handling layer works under the hood. Each of the routines in `file_mod` (see files in `src/io/file_handling/core`) takes a `file_handle` as its first argument. The `file_handle` contains a flag that indicates the format of the file it represents, and each of the routines in `file_mod` is basically a `SELECT` statement that defers to the correct implementation of the routine based on that flag. `file_handle`s are created in `file_open` – each file format implementation defines a list of recognised file extensions, and the appropriate file opening routine is deferred to by comparing the extension of the given file name to the recognised extensions for each file format.

To implement a new file format, an implementation of each of the routines in `file_mod` must first be provided (the provided implementations for ASCII and NetCDF formats should be used as examples). A new `CASE` deferring to the new implementation should then be added to the `SELECT` statement in each of the routines in `file_mod`. The recognised file extensions for the new format should also be added to the checks in `file_open` to allow the new the file opening routine to be called.

Implementations of these routines for ASCII and NetCDF file formats are given in `driver_ascii` (see `src/io/file_handling/core/drivers/ascii`) and `driver_ncdf` (see `src/io/file_handling/core/drivers/ncdf`) respectively – these should be used as examples of how to implement a file format. These two file formats suffer from opposite problems when implementing the concepts of dimensions, variables and records. For NetCDF, the concepts of dimensions and variables already exist, but the idea of a record has to be imposed. For ASCII, the concept of a record is a natural fit (think lines in a file), but the concepts of dimensions and variables have to be imposed. Between them, these implementations should provide sufficient examples of how to implement a file format.

7.2. Known limitations of the code

Limit to longest possible run

The longest possible run that can be attempted with JULES is approximately 100 years. A longer run should be split into smaller sections, with each later section starting from the final dump of the previous section. This restriction on run length arises because some of the time variables can become too large for the declared type of variable meaning that calculations return incorrect results and the program will probably crash. The size of each variable is in part affected by the compiler used, but a maximum run length of ~100 years appears to be a common case for 32-bit machines. Note that JULES uses the compiler's default KIND for each type of variable. Changes to the KIND of any variable would have to be propagated through the code.

Spin-up over short periods

The current code has not been tested with a spin-up cycle that is short in comparison to the period of any input data (e.g. a spin-up cycle of 1 day using prescribed vegetation data with a period of 10 days). The code will likely run but the evolution of the vegetation data may not be what was intended. However, it is unlikely that a user would want to try such a run.

8. Variables available for output

Variables that are available for output from JULES are listed in this section, separated according to their type. Note that all output is on the full model grid (even for variables defined on land points only). Any points on the grid for which a variable is not defined with be filled with a missing data value.

8.1. Variables that have a single value at each gridpoint (land and sea)

Table 50 List of output variables that have a single value at each gridpoint.

Name	Description
con_rain	Gridbox convective rainfall (kg m^{-2} s^{-1})
con_snow	Gridbox convective snowfall(kg m^{-2} s^{-1})
cosz	Cosine of the zenith angle (-)
diff_frac	Gridbox fraction of radiation that is diffuse (-)
ecan_gb	Gridbox mean evaporation from canopy/surface store (kg m^{-2} s^{-1})
ei_gb	Gridbox sublimation from lying snow or sea-ice (kg m^{-2} s^{-1})
esoil_gb	Gridbox surface evapotranspiration from soil moisture store (kg m^{-2} s^{-1})
fqw_gb	Gridbox moisture flux from surface (kg m^{-2} s^{-1})
ftl_gb	Gridbox surface sensible heat flux (W m^{-2})
land_albedo_1	Gridbox albedo for waveband 1 (direct beam visible)
land_albedo_2	Gridbox albedo for waveband 2 (diffuse visible)
land_albedo_3	Gridbox albedo for waveband 3 (direct beam NIR)
land_albedo_4	Gridbox albedo for waveband 4 (diffuse NIR)
latent_heat	Gridbox surface latent heat flux (W m^{-2})
ls_rain	Gridbox large-scale rainfall (kg m^{-2} s^{-1})
ls_snow	Gridbox large-scale snowfall (kg m^{-2} s^{-1})
lw_down	Gridbox surface downward LW radiation (W m^{-2})
precip	Gridbox precipitation rate (kg m^{-2} s^{-1})
pstar	Gridbox surface pressure (Pa)
q1p5m_gb	Gridbox specific humidity at 1.5m height (kg kg^{-1})
qw1	Gridbox specific humidity (total water content) (kg kg^{-1})
rainfall	Gridbox rainfall rate (kg m^{-2} s^{-1})
snomlt_surf_htf	Gridbox heat flux used for surface melting of snow (W m^{-2})
snowfall	Gridbox snowfall rate (kg m^{-2} s^{-1})
snow_mass_gb	Gridbox snowmass (kg m^{-2})
surf_ht_flux_gb	Gridbox net downward heat flux at surface over land and sea-ice fraction of gridbox (W m^{-2})
sw_down	Gridbox surface downward SW radiation (W m^{-2})

t1p5m_gb	Gridbox temperature at 1.5m height (K)
taux1	Gridbox westerly component of surface wind stress (N m^{-2})
tauy1	Gridbox southerly component of surface wind stress (N m^{-2})
tl1	Gridbox ice/liquid water temperature (K)
tstar_gb	Gridbox surface temperature (K)
u1	Gridbox westerly wind component (m s^{-1})
u10m	Gridbox westerly wind component at 10 m height (m s^{-1})
v1	Gridbox southerly wind component (m s^{-1})
v10m	Gridbox southerly wind component at 10m height (m s^{-1})
wind	Gridbox wind speed (m s^{-1})

8.2. Variables that have a single value at each land gridpoint

Table 51 List of output variables that have a single value at each land gridpoint.

Name	Description
albedo_land	Gridbox albedo (as used to calculate net shortwave radiation) (-)
canopy_gb	Gridbox canopy water content (kg m^{-2})
cs_gb	Gridbox total soil carbon (kg C m^{-2})
cv	Gridbox mean vegetation carbon (kg C m^{-2})
depth_frozen	Gridbox depth of frozen ground at surface (m)
depth_unfrozen	Gridbox depth of unfrozen ground at surface (m)
drain	Gridbox drainage at bottom of soil column (kg m^{-2} s^{-1})
elake	Gridbox mean evaporation from lakes (kg m^{-2}s^{-1})
emis_gb	Gridbox emissivity
fch4_wetl	Gridbox scaled methane flux from wetland fraction (10^{-9} kg C m^{-2}s^{-1})
fsat	Gridbox surface saturated fraction (-)
fsmc_gb	Gridbox soil moisture availability factor (beta) (-)
fwetl	Gridbox wetland fraction (-)
gpp_gb	Gridbox gross primary productivity (kg C m^{-2}s^{-1})
gs	Gridbox surface conductance to evaporation (m s^{-1})
hf_snow_melt	Gridbox snowmelt heat flux (W m^{-2})
land_index	Index (gridbox number) of land points
lice_index	Index (gridbox number) of land ice points
lit_c_mean	Gridbox mean carbon litter (kg C m^{-2} (360days)$^{-1}$)
lw_net	Gridbox surface net LW radiation (W m^{-2})
lw_up	Gridbox surface upward LW radiation (W m^{-2})

`npp_gb`	Gridbox net primary productivity (kg C m^{-2} s^{-1})
`qbase`	Gridbox baseflow (lateral subsurface runoff) (kg m^{-2} s^{-1})
`qbase_zw`	Gridbox baseflow (lateral subsurface runoff) from deep layer (kg m^{-2} s^{-1})
`rad_net`	Surface net radiation (W m^{-2})
`resp_p_gb`	Gridbox plant respiration (kg C m^{-2} s^{-1})
`resp_s_gb`	Gridbox total soil respiration (kg C m^{-2} s^{-1})
`resp_s_dr_out`	Gridbox mean soil respiration for driving TRIFFID (kg C m^{-2} $(360days)^{-1}$)
`runoff`	Gridbox runoff rate (kg m^{-2} s^{-1})
`sat_excess_roff`	Gridbox saturation excess runoff rate (kg m^{-2} s^{-1})
`smc_avail_top`	Gridbox available moisture in surface layer of depth given by `zsmc` (kg m^{-2})
`smc_avail_tot`	Gridbox available moisture in soil column (kg m^{-2})
`smc_tot`	Gridbox total soil moisture in column (kg m^{-2})
`snomlt_sub_htf`	Gridbox sub-canopy snowmelt heat flux (W m^{-2})
`snow_can_gb`	Gridbox snow on canopy (kg m^{-2})
`snow_depth_gb`	Gridbox depth of snow (m)
`snow_frac`	Gridbox snow-covered fraction of land points (-)
`snow_frac_alb`	Gridbox average weight given to snow for albedo (-)
`snow_grnd_gb`	Gridbox average snow beneath canopy (snow_grnd) (kg m^{-2})
`snow_ice_gb`	Gridbox frozen water in snowpack (kg m^{-2}) Only available if `nsmax`>0.
`snow_liq_gb`	Gridbox liquid water in snowpack (kg m^{-2}) Only available if `nsmax`>0.
`snow_melt_gb`	Gridbox rate of snowmelt (kg m^{-2} s^{-1})
`soil_index`	Index (gridbox number) of soil points
`sthzw`	Soil wetness in the deep (water table) layer (-)
`sub_surf_roff`	Gridbox sub-surface runoff (kg m^{-2} s^{-1})
`surf_roff`	Gridbox surface runoff (kg m^{-2} s^{-1})
`swet_liq_tot`	Gridbox unfrozen soil moisture as fraction of saturation (-)
`swet_tot`	Gridbox soil moisture as fraction of saturation (-)
`sw_net`	Gribox net shortwave radiation at the surface (W m^{-2})
`tfall`	Gridbox throughfall (kg m^{-2} s^{-1})
`trad`	Gridbox effective radiative temperature (K)
`zw`	Gridbox mean depth to water table (m)

8.3. Variables that have a value for each PFT at each land gridpoint

Table 52 List of output variables that have a single value for each PFT at each land gridpoint.

Name	Description
`c_veg`	PFT total carbon content of the vegetation (kg C m^{-2})
`canht`	PFT canopy height (m)
`flux_o3_stom`	PFT flux of O3 to stomata (mol m-2 s-1)
`fsmc`	PFT soil moisture availability factor (-)
`g_leaf`	PFT leaf turnover rate ($[360days]^{-1}$)
`g_leaf_day`	PFT mean leaf turnover rate for input to PHENOL ($[360days]^{-1}$)
`g_leaf_dr_out`	PFT mean leaf turnover rate for driving TRIFFID ($[360days]^{-1}$)
`g_leaf_phen`	PFT mean leaf turnover rate over phenology period($[360days]^{-1}$)
`gpp`	PFT gross primary productivity (kg C m^{-2} s^{-1})
`lai`	PFT leaf area index (-)
`lai_phen`	PFT leaf area index after phenology (-)
`lit_c`	PFT carbon litter (kg C m^{-2} $(360days)^{-1}$)
`npp_dr_out`	PFT mean NPP for driving TRIFFID (kg C m^{-2} $(360days)^{-1}$)
`npp`	PFT net primary productivity (kg C m^{-2} s^{-1})
`o3_exp_fac`	PFT ozone exposure factor
`resp_p`	PFT plant respiration (kg C m^{-2} s^{-1})
`resp_w_dr_out`	PFT mean wood respiration for driving TRIFFID (kg C m^{-2} $(360days)^{-1}$)
`resp_w`	PFT wood respiration (kg C m^{-2} s^{-1})

8.4. Variables that have a value for each tile at each land gridpoint

Table 53 List of output variables that have a single value for each tile at each land gridpoint.

Name	Description
`alb_tile_1`	Tile land albedo, waveband 1 (direct beam visible)
`alb_tile_2`	Tile land albedo, waveband 2 (diffuse visible)
`alb_tile_3`	Tile land albedo, waveband 3 (direct beam NIR)
`alb_tile_4`	Tile land albedo, waveband 4 (diffuse NIR)
`anthrop_heat`	Anthropogenic heat flux for each tile (W m^{-2})
`canopy`	Tile surface/canopy water for snow-free land tiles (kg m^{-2})
`catch`	Tile surface/canopy water capacity of snow-free land tiles (kg m^{-2})
`ecan`	Tile evaporation from canopy/surface store for snow-free land tiles (kg m^{-2} s^{-1})
`ei`	Tile sublimation from lying snow for land tiles (kg m^{-2} s^{-1})

emis	Tile emissivity
esoil	Tile surface evapotranspiration from soil moisture store for snow-free land tile (kg m^{-2} s^{-1})
fqw	Tile surface moisture flux for land tiles (kg m^{-2} s^{-1})
ftl	Tile surface sensible heat flux for land tiles (W m^{-2})
gc	Tile surface conductance to evaporation for land tiles(m s^{-1})
le	Tile surface latent heat flux for land tiles (W m^{-2})
nsnow	Tile number of snow layers (-)
q1p5m	Tile specific humidity at 1.5m over land tiles (kg kg^{-1})
rad_net_tile	Tile surface net radiation (W m^{-2})
rgrain	Tile snow surface grain size (μm)
snow_can_melt	Tile melt of snow on canopy (kg m^{-2} s^{-1})
snow_can	Tile snow on canopy (kg m^{-2})
snow_depth	Tile snow depth (m)
snow_grnd_rho	Tile bulk density of snow on ground (kg m^{-3})
snow_grnd	Tile snow on ground below canopy (kg m^{-2})
snow_ground	Tile snow on ground (snow_tile or snow_grnd) (kg m^{-2})
snow_ice_tile	Tile total frozen mass in snow on ground (kg m^{-2}) Only available if nsmax>0.
snow_liq_tile	Tile total liquid mass in snow on ground (kg m^{-2}) Only available if nsmax>0.
snow_mass	Tile lying snow (total) (kg m^{-2})
snow_melt	Tile snow melt rate (melt_tile) (kg m^{-2} s^{-1})
surf_ht_flux	Downward heat flux for each tile (W m^{-2})
surf_ht_store	C*(dT/dt) for each tile (W m^{-2})
t1p5m	Tile temperature at 1.5m over land tiles (K)
tstar	Tile surface temperature (K)
z0	Tile surface roughness (m)

8.5. Variables that have a value for each tile type at each land gridpoint

Table 54 List of output variables that have a single value for each tile type at each land gridpoint.

Name	Description
frac	Fractional cover of each surface type.
tile_index	Index (gridbox number) of land points with each surface type

8.6. Variables that have a value for each soil level at each land gridpoint

Table 55 List of output variables that have a single value for each soil level at each land gridpoint.

Name	Description
`b`	Brooks-Corey exponent for each soil layer (-)
`ext`	Extraction of water from each soil layer (kg m^{-2} s^{-1})
`hcap`	Soil heat capacity (J K^{-1} m^{-3}) for each soil layer
`hcon`	Soil thermal conductivity (W m^{-1} K^{-1}) for each soil layer
`satcon`	Saturated hydraulic conductivity (kg m^{-2} s^{-1}) for each soil layer
`sathh`	Saturated soil water pressure (m) for each soil layer
`smcl`	Moisture content of each soil layer (kg m^{-2})
`soil_wet`	Total moisture content of each soil layer, as fraction of saturation (-)
`sthf`	Frozen moisture content of each soil layer as a fraction of saturation (-)
`sthu`	Unfrozen moisture content of each soil layer as a fraction of saturation (-)
`t_soil`	Sub-surface temperature of each layer (K)
`sm_crit`	Volumetric moisture content at critical point for each soil layer (-)
`sm_sat`	Volumetric moisture content at saturation for each soil layer (-)
`sm_wilt`	Volumetric moisture content at wilting point for each soil layer (-)

8.7. Variables that have a value for each soil carbon pool at each land gridpoint

If `l_triffid` = TRUE, there are 4 soil carbon pools as in the RothC model (decomposable plant material, resistant plant material, biomass, humus). If `l_triffid` = FALSE, there is a single soil carbon pool.

Table 56 List of output variables that have a single value for each soil carbon pool at each land gridpoint.

Name	Description
`cs`	Carbon in each soil pool (kgC m^{-2})
`resp_s`	Respiration rate from each soil carbon pool (kgC m^{-2} s^{-1})

8.8. Variables that have a value for each snow layer on each tile at each land gridpoint

These variables are only available when `nsmax` > 0 (see Section 5.3). Snow layer variables are treated differently from other output variables, since they have two additional dimensions (`ntiles` and `nsmax`) - a separate output variable is used for each tile. Output for each tile must be requested separately in the `JULES_OUTPUT_PROFILE` namelist (see Section 5.17.2).

In Table 57, *<n>* should be replaced with a valid tile number to request output for that tile.

Table 57 List of output variables that have a single value for each snow layer at tile each land gridpoint.

Name	Description
`rgrainl_`*<n>*	Grain size in snow layers for each tile (µm)
`snow_ds_`*<n>*	Depth of each snow layer for each tile (m)
`snow_ice_`*<n>*	Mass of ice in each snow layer for each tile (kg m^{-2})
`snow_liq_`*<n>*	Mass of liquid water in each snow layer for each tile (kg m^{-2})
`tsnow_`*<n>*	Temperature of each snow layer (K)