Joint UK Land Environment Simulator (JULES)

Version 3.0

User Manual

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

Last revised by Matt.Pryor on 25 February 2011.

This version describes JULES v3.0.

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 2.2 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 3.0

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

1.2. 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.

1.3. 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.

1.3.1. Process descriptions

The main change is that a new multi-layer snow scheme is available (see `nsmax` in Section 6.2). 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.

1.3.2. 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 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).

1.4. 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.

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 two pieces of software:
- a Fortran 90 compiler with a pre-processor
- a version of the 'make' utility

It may also be desirable, but not essential, to have available the following software:
- the Fortran 90 netCDF[1] interface library

3.1. The make utility

The Makefile supplied with the JULES code should be compliant with most versions of make, but is only guaranteed to work with GNU Make[2] (also known as gmake), which is available on most Linux and UNIX systems and also for Windows. Once the Makefile is set up for the user's system, JULES is built simply by entering 'make' at the command prompt while in the directory containing the Makefile. This will compile all of the JULES source code, make a library libjules.a, and finally link the compiled source to create and executable file with a default name of jules.exe. To remove all the files created during the build process enter 'make clean' at the command prompt.

The make utility uses architecture- and compiler-specific variables that must be set by the user to the appropriate values for their system. These values may be set in the files Makefile.common.mk and Makefile.comp.*. (The user should not have to edit the file named Makefile.) There are two convenience options, COMPILER and BUILD, which should be passed to make from the command line to tell that program where the appropriate values should be taken from. The COMPILER option allows the user to define a list of compiler-specific variables (including compiler flags) without having to edit the Makefile. The BUILD option allows the user to build with sets of predefined flags for different situations, e.g. debugging. The Type and permitted values for each of these options are described in Table 1, and additional information is given in the comments at the head of Makefile.

The compiler-specific variables are specified in individual files named Makefile.comp.* for a handful of common compilers, e.g. Makefile.comp.sun. The list of tested compilers includes three (Intel, gfortran and G95) that can be downloaded for no cost via the URLs in Table 1 (certain conditions apply to these downloads). To use a compiler that is not listed, the user should replace the '@@' strings in the blank compiler file Makefile.comp.misc with values appropriate to their compiler and invoke make with the option COMPILER=misc.

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

Variable	Type and permitted values	Notes
COMPILER	Sun	Use options for Sun Studio compiler series (previously known as Workshop and Forte).

[1] The netCDF interface library can be downloaded for no cost from http://www.unidata.ucar.edu/software/netcdf/
[2] The GNU Make utility can be downloaded for no cost from http://www.gnu.org/software/make/

	Intel	Use options for Intel Fortran compiler for Linux, Windows and MacOS (http://www3.intel.com/cd/software/products/asmo-na/eng/compilers/284132.htm). Version 9.0 was used for testing of JULES2.0 and it was found that two lines in the source code had to be changed – find these and the suggested replacements by searching the code for "Intel".
	g95	Use options for G95 compiler (http://www.g95.org).
	Gfortran	Use options for the GNU fortran compiler (http://gcc.gnu.org/wiki/GFortran).
	Nag	Use options for NAGWare compiler.
	Pgf	Use options for Portland Group compiler.
	Misc	Use options for an unlisted compiler.
BUILD	Run	Default option; for normal compilation of JULES.
	Debug	Switch on compiler debug flags.
	Fast	Switch on compiler optimisation flags for faster execution.
CDFDUMMY	False	Use a precompiled netCDF library.
	True	Use the dummy netCDF library provided with JULES.

3.2. The JULES netCDF library

To build JULES, the user must also pass `make` some information about the netCDF interface library. If the user has access to a pre-compiled netCDF interface library, then they should pass `make` the options `CDF_LIB_PATH` and `CDF_MOD_PATH`. The values for these options are the directories in which the pre-compiled netCDF library (`libnc.a`) and Fortran 90 module files (those with `.mod` extension) are located respectively. This can be done also by editing the `Makefile` itself, but the recommended method is by specifying the variables as options when `make` is invoked, e.g., '`make CDF_LIB_PATH=$HOME/mynetcdf/lib CDF_MOD_PATH=$HOME/mynetcdf/mod`'.

If the user does not have access to a pre-compiled netCDF library, then JULES may be compiled by specifying '`CDFDUMMY=true`' when `make` is invoked rather than setting the `CDF_LIB_PATH` and `CDF_MOD_PATH` variables. This option compiles a set of dummy netCDF interface functions, which merely allows the rest of the JULES code to compile correctly and provides no functionality. **When this option is used JULES will neither read nor write netCDF files**. The user must then ensure that netCDF input/output options are not selected at any point in any JULES control file (described in Section 0) used with an executable produced using this option.
Example build linesTo build JULES using the normal Sun compiler options and link with a netCDF library:

```
make COMPILER=sun BUILD=run CDF_LIB_PATH=$HOME/mynetcdf/lib \
CDF_MOD_PATH=$HOME/mynetcdf/mod
```

To build JULES using the fast Intel compiler options and not link with a netCDF library:

```
make COMPILER=intel BUILD=fast CDFDUMMY=true
```

These command lines can become quite long and tedious to keep typing, so it's a good idea to set the list of frequently used ones as environment variables:

```
export JULESBUILD=”COMPILER=sun BUILD=run \
CDF_LIB_PATH=$HOME/mynetcdf/lib \
CDF_MOD_PATH=$HOME/mynetcdf/mod”

make $JULESBUILD
```

It is then possible to override options specified in that variable by adding revised ones at the end:

```
make $JULESBUILD BUILD=debug
```

3.3. Running JULES

A JULES executable is run by redirecting standard input to a file that contains all the information needed to describe a run, e.g.,
`jules.exe < run1.jin`

The format of this input file is described in Section 6, with some example runs described in Section 6.24.

The file extension “.jin” is meant to suggest “**J**ULES **in**put file”, but there is no need to use this or any other extension.

4. Overview of the JULES code

The general structure of the JULES source code, including the order in which routines are called, is illustrated below. For the sake of clarity, the full details are not shown here. In particular, the initialisation and output steps (subroutines `init` and `output`) can call several routines. The focus below is on the calling order for land points (rather than sea or sea-ice).

```
jules--|
       |--init--|
       |        |--init calls various initialisation routines
       |
 (top of timestep loop)
       |--imogen_update_clim (if IMOGEN is on)
       |
       |--drive_update
       |
       |--veg_update
       |
       |--control---|
       |            |--zenith
       |            |
       |            |--tile_albedo--|
       |            |               |--albpft
       |            |               |--albsnow
       |            |               |--canyonalb (MORUSES)--|
       |            |               |                       |--matinv
       |            |
       |            |--generate_anthropogenic_heat
       |            |
       |            |--sf_expl--|
       |            |           |--tilepts
       |            |           |--physiol--|
       |            |           |           |--albpft
       |            |           |           |--root_frac
       |            |           |           |--smc_ext
       |            |           |           |--raero
       |            |           |           |--sf_stom--|
       |            |           |           |           |--qsat
       |            |           |           |           |--leaf_limits
       |            |           |           |           |--leaf
       |            |           |           |--soil_evap
       |            |           |           |--leaf_lit
       |            |           |           |--cancap
       |            |           |           |--urbanemis (MORUSES)--|
       |            |           |           |                       |--matinv

       |            |           |           |--microbe
       |            |           |
       |            |           |--heat_con
       |            |           |--snowtherm
       |            |           |--hcons_snow
       |            |           |
       |            |           |--sf_exch--|
       |            |                       |--elevate--|
       |            |                       |           |--dewpnt
       |            |                       |           |--qsat
       |            |                       |--qsat_mix
       |            |                       |--urbanz0 (MORUSES)--|
       |            |                       |                     |--get_us
```

```
|            |                        |--sf_orog
|            |                        |--sf_resist
|            |                        |--sf_rib
|            |                        |--sf_orog
|            |                        |--fcdch----|
|            |                        |           |--phi_m_h
|            |                        |--sf_resist
|            |                        |--sf_flux
|            |                        |--stdev1
|            |                        |--sfl_int---|
|            |                        |            |--phi_m_h
|            |                        |--sf_orog_gb
|            |                        |--sf_aero--|
|            |                                    |--dustresb--|
|            |                                                 |--vgrav
|            |--sf_impl--|
|            |           |--im_sf_pt
|            |           |--sf_evap
|            |           |--sf_melt
|            |           |--screen_tq--|
|            |                         |--qsat
|            |
|            |--snow--|
|            |        |--canopysnow
|            |        |--layersnow
|            |        |--snowtherm
|            |        |--snowpack--|
|            |        |            |--tridag
|            |        |--snowgrain
|            |        |--compactsnow
|            |        |--relayersnow--|
|            |                        |--layersnow
|            |
|            |--hydrol---|
|            |           |--surf_hyd--|
|            |           |            |--frunoff
|            |           |            |--sieve
|            |           |            |--pdm
|            |           |--calc_baseflow
|            |           |--soil_hyd--|
|            |           |            |--hyd_con_ic--|
|            |           |            |              |--hyd_con_ch/vg
|            |           |            |--darcy_ic--|
|            |           |            |            |--darcy_ch/vg--|
|            |           |            |                             hyd_con_ch/vg
|            |           |            |--gauss
|            |           |            |--calc_zw
|            |           |--soil_htc--|
|            |           |            |--heat_con
|            |           |            |--gauss
|            |           |--ice_htc
|            |           |--soilmc
|            |           |--soilt
|            |           |--ch4_wetl
|            |
|            |--sice_htf
|            |
|            |--veg2--|
|            |        |--tilepts
|            |        |--phenol
|            |        |--triffid--|
```

```
      |            |        |           |--vegcarb--|
      |            |        |           |           |--growth
      |            |        |           |--lotka--|
      |            |        |           |         |--  compete
      |            |        |           |
      |            |        |           |--soilcarb--|
      |            |        |           |            |--dpm_rpm
      |            |        |           |            |--decay
      |            |        |--tilepts
      |            |        |--sparm--|
      |            |                  |--pft_sparm
      |            |--veg1--|
      |                     |--tilepts
      |                     |--phenol
      |                     |--sparm--|
      |                               |--pft_sparm
      |--imogen_update_carb (if IMOGEN is on)
      |--output
      |--new_time--|
      |            |--spin_check
      |
(bottom of timestep loop)
      |
      |--jules_final
```

5. File formats and the JULES grid

5.1. Overview of file formats

JULES aims to support input and output in three formats: ASCII, netCDF and a generic binary format (simply called ‘binary’ below). The implementation of netCDF input is fairly limited, in that only certain dimension names are allowed (see Section 5.2.2). Input can also be read from many PP files (a format used by the UK Met Office). The binary and netCDF files are compatible with the GrADS[3] package, amongst others. A run control file might indicate that data are to be read from several files, using one or more of these file formats. For example, soil data might be in an ASCII file, while meteorological driving data are in netCDF files.

A “self-describing file” (SDF) is one in a format that contains metadata describing the contents of the file. For JULES, only a netCDF file is presently considered to be a SDF. Minimal use is made of any metadata contained within a file, including SDFs and PP files. For example, a SDF might contain data that describes the grid or the times of data, but these are not used by JULES. Instead, this information is provided via the run control file and all input data must be provided on the same grid.

For all non-SDF files, the data model is based on that used by GrADS. Each variable is viewed as being 4-dimensional in (x, y, z, t) on a regular grid. Although we will talk of x and y in terms of West-East and South-North compass directions, in general the grid can be any rectilinear grid, with West-East being replaced by “left to right”. x varies in the direction from West to East, y varies from South to North (this default can be changed), and z varies from bottom to top. All variables in any one file must have the same grid size in x and y (i.e. all variables are on a grid of `nx*ny` points), and have a value at all times (although that value could indicate a missing datum). The data are stored as a series of xy slices, with x varying fastest, then y, then z, and t varying slowest. For example, say we have a file with two variables (A and B) on a grid with `nx=2`, `ny=2`. A has `nz=1`, and B has `nz=2`. In the JULES/GrADS model, the data must be stored in the input file in the order:

```
A(x=1,y=1,z=1,t=1)   # 1st xy plane of A at t=1
A(x=2,y=1,z=1,t=1)
A(x=1,y=2,z=1,t=1)
A(x=2,y=2,z=1,t=1)
B(x=1,y=1,z=1,t=1)   # 1st xy plane of B at t=1
B(x=2,y=1,z=1,t=1)
B(x=1,y=2,z=1,t=1)
B(x=2,y=2,z=1,t=1)
B(x=1,y=1,z=2,t=1)   # 2nd xy plane of B at t=1
B(x=2,y=1,z=2,t=1)
B(x=1,y=2,z=2,t=1)
B(x=2,y=2,z=2,t=1)
A(x=1,y=1,z=1,t=2)   # 1st xy plane of A at t=2
A(x=2,y=1,z=1,t=2)
… etc …
```

[3] The GrADS software can be downloaded for no cost from http://grads.iges.org/grads/gadoc/index.html

For clarity, this example has shown each datum on a separate line, but in fact any number of data within a single field (see below) can be on the same line.

A data "field" is considered to be a single x-y plane of data (i.e., `nx*ny` values). Header records can be present at the start of a file, at the start of each time within the file, and at the start of each field.

Note that this means that JULES reads and writes data in terms of 'maps' (all values of one field, then all values of another field), rather than using a 'point-by-point' data model (all fields for one point, then all fields for another point).

A related concept used in JULES, is that of the point number in input or output files. This is used to select individual points from a larger grid. The point number runs from 1 at the gridpoint in the SW corner of the grid, across rows (so the bottommost row contains gridpoints 1 to `nx`), and then from South to North up the grid. Examples and further discussion of JULES grids can be found in Section 6.4.

5.2. Describing the format of an input file

Variables that describe how data are arranged in files are used in several sections later in this document. These variables are summarised in Table 2. Often the information that JULES will read and use from the control file depends on the file format of any one data file. The information required for an ASCII, binary or PP file is generally fairly similar, while netCDF files are rather different.

Table 2 Frequently used control file options

Variable name	Type	Notes
`readFile`	Logical	Switch that indicates source of data. TRUE: data are read from a named, external file FALSE: data are read from the run control file
`fileFormat`	Character	Flag indicating the file format. Case sensitive. Only used if `readFile=.TRUE.` `'asc'`: ASCII `'bin'`: generic binary (including GrADS) `'nc'`: netCDF `'pp'`: PP format

5.2.1. ASCII or binary input files

If `fileFormat='asc', 'bin' or 'pp'` or `'pp'` some or all of the following information is read from a section that starts with the tag '`>ASCBIN`'. Exactly what information is needed varies between cases (for example, it is assumed that there is a single time "level" in a file of soil properties, so `nheaderTime` is not needed).

Table 3 Options used to specify the reading of ASCII, binary and PP format files.

Variable name	Type	Notes

`nheaderFile`	Integer	The number of header records at the top of a file. For an ASCII file, a header record is a line in the file. For a binary file, a header record is an individual word or record (e.g. a single 'real' value). Not used for a PP file.
`nheaderTime`	Integer	The number of header records that precedes the data for each time level within a file. Not used for a PP file.
`nheaderField`	Integer	The number of header records that precedes each field (x-y plane) of data. Not used for a PP file.
`fieldNumber`	Integer	This is used to locate a given field (xy plane) within all the fields available at each time level. If there are `nFieldFile` fields of data at each time level, and `fieldNumber=2` for a particular variable, the second field of data is used for this variable.

Blank lines between fields in an ASCII input file can cause the code to read the wrong data, and should be avoided. If blank lines are present between fields, they should be interpreted as header lines.

There are restrictions on what PP files JULES can read. Each field must have no trailing "extra data" (i.e. `header(20)` must be zero). It is also assumed that the data are ordered as in the JULES/GrADS model outlined above (so, for example, we do NOT have all times of field 1, then all times of field 2), so that the required data can be found without using the information contained in the field headers. The headers are used to check that the size of the field and the STASH code are as expected. The STASH code for each variable is currently hardwired in the code. At the time of writing the PP-reading code has no known bugs, but it has been used much less than other options, so any more obscure bugs might not have been triggered.

5.2.1.1. An example ASCII input file

Table 4 shows part of an example ASCII file that could be read by JULES, with `nheaderFile`=2, `nheaderTime`=1, `nheaderField=1`. The size of the input grid is assumed to be `nxIn`=3, `nyIn`=2. There are 2 variables, A which has a single level, and B which has 2 levels, giving a total of 3 fields per time. Annotation after any "!" (and shown in *italics*) would NOT be present in the actual file. The data are shown on 2 lines per field, but this is not important – `nx*ny` values will be read however they are presented.

Table 4 Part of an example ASCII file that could be read by JULES.

```
This file contains example data.                        !  1st file header
There are 2 variables, the 2nd with 2 levels.           !  2nd file header
Time level 1.                                           !  header for time=1
Variable A                                              !  header for 1st field
12.0   15.6    17.1                                     !  data for A at t=1
-1.0   23.9    53.2
Variable B, level 1                                     !  header for 2nd field
22.0   25.6    12.1                                     !  data for B at t=1, 1st level
-1.0   22.9     23.2
Variable B, level 2                                     !  header for 3rd field
32.0   11.6    12.1                                     !  data for B at t=1, 2nd level
-9.1   72.9     43.7
Time level 2.                                           !   header for time=2
Variable A                                              !   header for 1st field
9.2     67.3    -7.6                                    !   data for A at t=2
11.5   23.9    -8.3
Variable B, level 1                                     !   header for 2nd field
---- rest of file not shown ---
```

5.2.2. netCDF input files

If `fileFormat='nc'`, the required information is read from a section that starts with the tag '`>NC`'. The only information that is required is the name of the netCDF variable.

To be used with JULES, a netCDF file must meet certain requirements and be in the format of one of several "types" which are summarised in Table 6. The types are used to summarise the names and order of the dimensions of variables in the file (see Table 6). The type of netCDF files to be read in a particular run is specified by the variable `ncType` (see Section 6.2), except that the type of meteorological data is specified by `ncTypeDrive` (Section 6.18). The provision for netCDF input and the creation of these types have been added in a rather ad hoc manner as need has arisen. Provision for netCDF input will likely be improved in a future version of JULES. In general there is more flexibility for reading driving (meteorological) data from netCDf files. If other types of input are in netCDf files that do not conform to the requirements, they need to be rewritten with the required dimension names, or converted to another file format. Another alternative is that the user can modify the JULES code – it is fairly easy to add another netCDF "type" (most of the relevant code is in `jules_netcdf.f90`).

Table 5 Recognised types of netCDF input file

Type name	Notes
gswp2	Refers to the Global Soil Wetness Project 2 (http://www.iges.org/gswp2 - although data are no longer available from that site).
pilps2e	Can only be used for meteorological data. The PILPS2e experiment is described in Bowling, L.C. and co-authors, 2003, *Simulation of high latitude hydrological processes in the Torne-Kalix basin: PILPS Phase 2(e), 1: Experiment design and summary intercomparisons*, Global and Planetary Change, 38 (1-2): 1-30. The data are not widely available.
princet	Can only be used for meteorological data. These data from Princeton University are described in Sheffield, J., G.Goteti and E.F.Wood, 2006, *Development of a 50-yr high-resolution global dataset of meteorological forcings forland surface modelling*, J.Climate, 19: 3088-3111, and can be downloaded from http://hydrology.princeton.edu/data.pgf.php.
tseries	Can only be used for meteorological data. A simple format for time series at a single point.
watch	The Water and Global Change project (WATCH; www.eu-watch.org) is an EU FP6 project which is producing meteorological data for model input, amongst other aims. These data are not yet widely available.

Table 6 Dimensions in netCDF input files

Related section of run control file (see Section 6).	Allowable values of `ncType` (`ncTypeDrive` for `INIT_DRIVE`)	Required dimension names (case and order are important)
INIT_LAND, INIT_LATLON	gswp2	Land
	watch	land
INIT_FRAC	gswp2	Land, Psuedo,Time[4]
	watch	land, pseudo
INIT_SOIL	gswp2	Land Note that vertically-varying soil data cannot be read from a netCDF file and the code will stop at any attempt to do so.
	watch	land
INIT_HGT	gswp2	Land, Psuedo
	watch	land, pseudo
INIT_TOP	gswp2	Land
	watch	land
INIT_VEG_VARY	gswp2	Land, Psuedo, Time
	watch	land, pseudo, tstep
INIT_URBAN	gswp2	Land
	watch	land
INIT_AGRIC	gswp2	Land
	watch	land
INIT_DRIVE	gswp2, watch	land, tstep
	pilps2e	x, y, tstep
	princet	longitude, latitude, z, time
	series	time
INIT_IC	gswp2	Land, Psuedo, Soil Note that these dimensions are insufficient to cope with all possible variables. If an attempt is made to read another kind of variable, the code will report an error and stop.

[4] Note the typographical error for files of type ‘gswp2’– Psuedo rather than Pseudo! This crept in when those files were created and files of this type continue to have to use Psuedo. Consider using files of type ‘watch’ instead.

6. The JULES control file

6.1. Introduction

Each run of the JULES code is controlled by a text file that is called the “run control file”. Broadly speaking, the run control file holds three types of information:

- the general details of the run, such as start and end dates
- the values for parameters of the model, such as albedo
- the specification of the required output

The JULES code is designed to be moderately flexible, in that there are switches that allow the user to select between different configurations, and it can accommodate input data in several different file formats. This flexibility means that the run control file may be relatively long and the user has to check that all values are set correctly. The documentation below aims to help the user in this task. Example input files can be found as described in Section 6.24.

The run control file has a particular format, in that the lines must be in a particular order and must contain various headers. The file is read by various routines arranged under the subroutine INIT, using FORTRAN list-directed input [i.e. the format is given as “*” in a `READ` statement of the form `READ(unit,*)`]. The JULES executable is run with standard input redirected to this control file, e.g. `jules.exe < control_file.jin`. The use of list-directed input means that there may be more than one arrangement of input values that can be read by the code – for example a single line with 10 values or 2 lines with 5 values each. Repeated numerical values can often be specified using the “*” notation (e.g. 100 values of 1.0 can be entered as `100*1.0`), which can sometimes be useful in specifying a large but constant field.

“Tags” are used to indicate the start of each section, and allow the code to skip directly to this point ignoring any intermediate lines. Each tag is of the form,

```
>SECTION_NAME
```

and must be included exactly as in the example run control files, using capital letters and with no space before or after the initial >. These section tags are listed in Table 7.

Table 7 Sections in a JULES control file.

Section name	Description	Described in manual section
`INIT_OPTS`	General model options.	6.2
`INIT_TIME`	Start and end times for simulation, timestep lengths, spin up.	6.3

`INIT_GRID`
`INIT_LAND`
`INIT_LATLON`
	Set up the model grid.	6.4

`INIT_FRAC`	Set gridbox tile fractional coverage options.	6.5
`INIT_SOIL`	Set model soil parameters.	6.6
`INIT_TOP`	Set values for a TOPMODEL-type parameterisation of runoff.	6.7

INIT_PDM	Set parameters for a PDM-type parameterisation of surface runoff.	6.8
INIT_HGT	Set the relative elevation of each tile.	6.9
INIT_VEG_PFT	Set uniform parameters for vegetation tiles.	6.10
INIT_VEG_VARY	Set parameters for vegetation tiles that vary in either space or time.	6.11
INIT_NONVEG	Set parameters for non-vegetation tiles.	6.12
INIT_URBAN	Urban model configuration, geometry & material characteristics	6.13
INIT_SNOW	Set snow related parameters.	6.14
INIT_TRIF	Set parameters for TRIFFID dynamic vegetation model.	6.15
INIT_AGRIC	Set fraction of each gridbox that is agriculture for use with TRIFFID.	6.16
INIT_MISC	Set miscellaneous carbon-cycle parameters.	6.17
INIT_IMOGEN	Set IMOGEN options	6.18
INIT_DRIVE	Set input driving data options.	6.19
INIT_IC	Set initial conditions of all prognostic variables.	6.20
INIT_OUT	Set options for model output.	6.21
	General output options	6.21.1
	NEWPROF: Set up an output profile.	6.21.2

The user can annotate the run control file, for example to add comments, but these must not interfere with the reading of the rest of the file. Depending upon the details of the run, there are various locations in which it is "safe" to include annotation, but the only really safe location is on the lines immediately preceding a "tag" (described above). Annotation can also often be placed on the same line as the data at the end of any data field (i.e. so that the code reads the values required and will not see the annotation).

Values of character variables, such as file names, should be enclosed within quotation marks (either single ' ' or double " "). Character variables have a maximum length specified in the code, which are sometimes given in this documentation, e.g. character*8 indicates a variable of length 8. Logical values can be entered in any of the formats understood by FORTRAN, e.g. T, true or `.TRUE.` may all be used to represent true. In the sections below, the sizes of certain arrays are indicated using brackets: e.g. `myArray(1:20)` requires values for the 20 elements numbered 1 to 20.

Although a spatial field can be read from the run control file, in practice this becomes unwieldy for large grids, and most spatial fields are likely to be stored in separate files, the names of which can be listed in the run control file.

In the following sections, the first column lists the variables that are to be read from a line, and subsequent columns give the type and a brief description of each variable. The variable names given are generally those used to declare the corresponding FORTRAN variables (except where the code uses temporary workspace and a more meaningful variable name is given in this documentation).

6.2. `INIT_OPTS`: General model options

This section starts with the tag `>INIT_OPTS`.

```
>INIT_OPTS

npft, nnvg
l_aggregate
pftName(1:npft)
nvgname(1:nnvg)

nxIn, nyIn
sm_levels
nsmax
can_model
can_rad_mod, ilayers
l_cosz, l_spec_albedo
l_phenol, l_triffid, l_veg_compete, l_trif_eq
l_top,l_pdm
l_anthrop_heat_src, l_moruses
l_o3_damage
l_imogen
l_epot_corr, l_snowdep_surf

i_scrn_t_diag

yrevIn
ncType
echo
print_step
```

Table 8 Description of variables in INIT_OPTS section.

Variable name	Type and permitted values	Notes
`Npft`	integer >=1	The number of plant functional types to be modelled.
`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 the section `INIT_FRAC` (Section 6.5).

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. Generally l_aggregate=.FALSE. is preferred, TRUE can be used to reduce the computational cost.
pftName(1:npft)	character array	Names of PFTs. When JULES looks for parameter values for the PFTs, it looks for these names.
nvgName(1:nnvg)	character array Must include 'soil'.	Names of non-vegetation surface types. When JULES looks for parameter values for the surface types, it looks for these names.
nxIn	integer >=1	The number of columns of data in the input grid (see further discussion of the grid in Section 6.4).
nyIn	integer >=1	The number of rows of data in the input grid.
		The total number of points in the input grid is thus nxIn*nyIn. If the input data consists of a single point, nxIn=nyIn=1. A vector of points is specified by setting nyIn=1. Although the notation may suggest a regular, rectangular grid, the model can be run at any number of arbitrary locations, the most likely way of doing so being to set nyIn=1, nxIn=number of points.
sm_levels	integer >=1	Number of soil layers. A value of 4 is often used.
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. The minimum depth of each layer is set in Section 6.14.

`can_model`	integer 1, 2, 3 or 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.

`can_rad_mod`	integer 1, 2 3, 4 or 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. This includes, i) sunfleck penetration though the canopy, ii) division of sunlit and shaded leaves within each canoy level and iii) 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 (in prep). References: Jogireddy, V.R. et al., 2006, Hadley Centre technical note 63. Available from: http://www.metoffice.gov.uk/publications/HCTN. Sellers, P. et al., 1992, Remote Sens. Environ., 42: 187-216. Mercado et al. 1997, Tellus B, 59, 553–565
`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.

`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.
`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.
`l_phenol`	logical	Switch for vegetation phenology model. TRUE: use phenology model. FALSE: do not use phenology model.
`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.
`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
`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.
`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.

`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.
`l_anthrop_heat_src`	logical	Switch for inclusion of anthropogenic contribution to the surface heat flux from urban tiles. The relevant code is found in subroutine generate_anthropogenic_heat. TRUE: add anthropogenic effect FALSE: no effect
`l_moruses`	logical	Switch for turning on MORUSES. Configuration of and urban parameters required for MORUSES are set in `INIT_URBAN` (Section 6.13) **TRUE: use MORUSES parametrisations**. Requires `nvgName` types `'urban_roof'` and `'urban_canyon'`[5] **FALSE: do not use MORUSES parametrisations.** Use urban tile parameters, set in `INIT_NONVEG` (Section 6.12), 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

[5] Both the two tile schemes, URBAN-2T & MORUSES, will also run with the `'urban'` surface type as the code converts this to the `'urban_canyon'` type itself as long as the `'urban_roof'` tile is present. However, they will fail to run if both `'urban'` and `'urban_canyon'` are present. When entering the urban fraction data the **total** urban fraction should be entered in the `'urban_canyon'` or `'urban'` tile, whichever is named.

`l_o3_damage`	logical	Switch for ozone damage. TRUE: Ozone damage is on. Ozone concentration in ppb must be supplied as a driving variable FALSE: No effect
`l_imogen`	logical	Switch for IMOGEN TRUE: IMOGEN is used to generate driving data FALSE: No effect
`l_epot_corr`	logical	TRUE: use correction to the calculation of potential evaporation FALSE: No effect
`l_snowdep_surf`	logical	TRUE: use equivalent canopy snow depth for surface calculations on tiles with a snow canopy FALSE: no effect
`i_scrn_t_diag`	Integer 0 or 1	Switch controlling method for diagnosing screen temperature. 0: use surface similarity theory. This is the default and acceptable for most users. 1: use surface similarity theory but allow decoupling in very stable conditions based on the quasi-equilibrium radiative solution
`yrevIn`	logical	Switch indicating if the order of the rows in the input data is not the JULES standard. TRUE: Input data are arranged in North to South order (i.e. first data are from northernmost row). FALSE: Input data are arranged in South to North order (the JULES standard). Note that this does not affect how JULES numbers points on its internal grids – within JULES the numbering always runs from South to North. This switch applies to all input files.
`ncType`	character	Indicates the type (format) of any netCDF input files (see Section 5.2.2). This does not refer to files for meteorological data which are covered in Section 6.18.
`echo`	logical	Switch controlling output of messages to standard output (e.g. screen). TRUE: print messages to screen. This will print the values of parameters, and also print messages when files are opened or closed. This is useful while checking that a run is correctly set up, but can result in a large volume of data if the model grid is large. FALSE: suppress printing of most messages to screen
`print_step`	integer >=1	The number of timesteps in between the printing of timestep information. Every `print_step` timesteps, the model prints the current timestep number and date to standard output. While this can be a useful way to follow the progress of a model integration, frequent messages can generate a large amount of unnecessary output during long integrations.

6.3. `INIT_TIME`: Date and time information

This section sets the start and end time of the run and can also be used to specify a spin-up procedure. It starts with the tag `>INIT_TIME`.

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

```
>INIT_TIME

timestep
dateMainRun(1), timeRun(1)
dateMainRun(2), timeRun(2)

l_360
phenol_period, triffid_period

dateSpin(1:2), nspin
spinFail
>VARS
spinVarName(1),spinTolPercent(1), spinTol(1)
--- Repeat for each variable. ---
>ENDVARS
```

Table 9 Description of variables in the `INIT_TIME` section

Variable name	Type and permitted values	Notes
`timestep`	integer >=1	Timestep length (seconds). A typical timestep is 30 to 60 minutes. If the timestep is too long, the model becomes numerically unstable.
`dateMainRun(1:2)` `timeRun(1:2)`	integer array, character*8 array	These specify 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. For simplicity, the same times of day are used for both the spin-up and main periods. The main run starts at `timeRun(1)` on `dateMainRun(1)` and ends at `timeRun(2)` on `dateMainRun(2)`. Dates should be given in format yyyymmdd. All dates must be >0. Times should be given in format hh:mm:ss. **It is recommended that all times entered in JULES use Greenwich Mean Time (GMT or UTC), not local time. The use of GMT is essential if certain options are set (`l_cosz`=TRUE)** - but see

		6.3.1 for a possible, if not recommended, use of local time!
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.
phenol_period	integer >=1	Period for calls to phenology model (days). Only relevant if l_phenol=TRUE.
triffid_period	integer >=1	Period for calls to TRIFFID model (days). Only relevant if one of L_TRIFFID or L_TRIF_EQ is TRUE.
dateSpin(1:2)	integer array	The dates for the spin-up period, in the format yyyymmdd. Elements 1 and 2 are the start and end dates respectively. The spin-up phase of the integration must be over times that either, • immediately precede the main run. In this case the spin-up phase is from timeRun(1) on dateSpin(1) to timeRun(1) on dateSpin(2) [where dateSpin(2) equals dateMainRun(1)] OR • are the same as those for the main run. In this case the spin-up phase is from timeRun(1) on dateMainRun(1) to timeRun(2) on dateMainRun(2). Examples are given below.
nspin	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 nspin 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 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.
spinFail	logical	Switch controlling behaviour at the end of spin up period, if the model has not passed the spin-up test. Only used if nspin>0. TRUE: End the run if model has not spun up. FALSE: Continue the run.
If nspin>0, details of the variables used to assess the spin up are looked for between the tags >VARS and >ENDVARS. Up to two variables can be listed.		
spinVarName	character Acceptable values are: ‘smcl’	The name of a variable to be used to determine if the model has spun up. Spin up can be assessed in terms of soil temperature and soil moisture. ‘smcl’: moisture content of each soil layer (kg m^{-2})

	't_soil'	't_soil': temperature of each soil layer (K)
spinTolPercent	logical	Switch indicating whether the tolerance for this variable is expressed as a percentage. TRUE: tolerance is a percentage FALSE: tolerance is an absolute value
spinTol	real	Tolerance for spin up of this variable. For each spin-up variable, this is the maximum change over a repetition of spin up that is allowed if the model is to be considered as spun-up. If the absolute value of the change (the percentage change if spinTolPercent = TRUE) is less than or equal to spinTol, the variable is considered to have spun up. For example, spinTol=0.1 means that the variable in question must change by less than 0.1 over a cycle of spin up if it is to be considered spun up. See notes below on using a negative tolerance to prescribe the number of cycles that are attempted. 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 15 Jan 2005 to 15 Jan 2006, every time the model gets to 15 Jan 2006 the spin-up variables are compared with their value at the end of the previous cycle.

6.3.1. Note on time convention and 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.

6.3.2. Examples of dates and times

1. A run without spin up

```
19970101, '00:00:00'    !  start date and time
19990101, '01:00:00'    !  end date and time
19970101, 19970102, 0   !  dateSpin, nspin
```

This specifies a run from midnight on 1st January 1997 until 01:00 GMT on 1st January 1999. nspin=0 means there is no spin up.

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

```
19970101, '00:00:00'    !  start date and time for main run
19990101, '01:00:00'    !  end date and time for main run
19960101, 19970101, 5   !  dateSpin, nspin
```

This specifies a spin-up period from midnight on 1st January 1996 to midnight on 1st January 1997 (the time of day is taken from the first line). 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.

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

```
19970101, '00:00:00'    !  start date and time for main run
19990101, '01:00:00'    !  end date and time for main run
19970101, 19980101, 5   !  dateSpin, nspin
```

This specifies a spin-up period from midnight on 1st January 1997 to midnight on 1st January 1998 (the time of day is taken from the first line). 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.

4. Example of specifying requirements for spin up

```
T                !  terminate run if spin-up fails (T,F)
smcl   F  1.0    !  spinVarName,spinTolPercent,spinTol
t_soil T  0.1    !
```

The first value, `spinFail`=TRUE, means that if the spin-up has not "converged" after `nspin` cycles, the run will end. Convergence is measured using 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%.

6.3.3. Notes on spin up

Note that 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 (TRIFFID; the equilibrium mode of TRIFFID is designed to spin up TRIFFID) or prognostic soil carbon.

During the spin-up phase of a run, the JULES code 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 over 1 Jan 2005 to 31 Dec 2005. At or near the end of 31 Dec 2005 during the spin up, the driving data will start to adjust to the values for 1 Jan 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. 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 over 1 Jan 2005 to 31 Dec 2005. At or near the end of 31 Dec 2005 during the spin up, the driving data will start to adjust to the values for 1 Jan 2005, which could be very different from conditions on 31 Dec 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 1 Jan 2005.

Although `nspin` 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 `spinTol`<0 (remember that `spinTol` is compared with the *absolute* change over a cycle) and setting `spinFail=FALSE` so that the integration continues when spin up is judged to have failed after `nspin` 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 `spinTol`<0, setting `spinFail=TRUE` so that the integration stops when spin up is judged to have failed, and setting `dumpFreq` (see Section 6.21.1) to any value that writes a final dump. The final state of the model, after `nspin` 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.

6.4. Grid description

The process of setting up the model grid involves three parts of the run control file: INIT_GRID, INIT_LAND and INIT_LATLON.

INIT_GRID is used to select how the model grid will be specified, e.g. all points within a given range of latitude and longitude.
INIT_LAND is used to set a land/sea mask.
INIT_LATLON specifies the latitude and longitude of each point.
These three sections are then followed by the DATA_POINTS, DATA_LAND and DATA_LATLON sections which provide input data (if that is to come from the run control file).

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 a subset of the input grid.

The JULES grid is a rectangle of size nx*ny points, including the special case of ny=1 when the grid is a vector of points. The points to be modelled may be selected from a larger input grid, by specifying one or more of (1) a list of point numbers (2) a range of latitude and longitude (3) that only land points (really points at which a mask is >0) are to be selected. The grid may contain both land and sea points, but at present JULES is only deals correctly with land points, so results for sea points will be meaningless and are therefore better omitted as described later. A vector of points can be used to select locations that are not adjacent in the real world - for example, one might only want to run the model at locations within a catchment for which observations are available. In this case although the model could be run on a grid that included the whole catchment, it is more efficient to run only at the selected points.

6.4.1. INIT_GRID: Setting up the grid

```
>INIT_GRID

pointsList, coord, coordLL
landOnly
subArea, subAreaLatLon
xcoord(1:2),ycoord(1:2)
npoints
readFilePoints
fileNamePoints
```

Table 10 Description of variables in the INIT_GRID section

Variable name	Type and permitted values	Notes
pointsList	logical	Switch indicating whether the model grid is to be specified as a list of points.

		TRUE: Points to be modelled are selected from the input grid via a list provided by the user. In this case, the points to be modelled are selected via a list of point numbers (or coordinate pairs, see below). **FALSE: All points in the input grid are modelled – subject to elimination by `subArea` or `landOnly` (see below).** The value of `npoints` (q.v.) is set by the model, and equals the number of points that are modelled.
`coord`	logical	Switch indicating if a list of points is given as co-ordinate pairs. Only used if `pointsList`=TRUE. TRUE: The list of points will be given as co-ordinate pairs. FALSE: The list of points will be given in terms of single per point, describing the location in the grid.
`coordLL`	logical	Switch indicating if co-ordinate pairs are (x,y) or (longitude,latitude). Only used if `coord`=TRUE. TRUE: Co-ordinates are (longitude,latitude). FALSE: Co-ordinates are (x,y) in the input grid.
`landOnly`	logical	Switch indicating if only land points are to be modelled. If `pointsList=TRUE`, `landOnly` must be FALSE. **TRUE: Only land points are modelled. Sea points are excluded from the model grid.** More correctly, only points with `flandg (see later)` >0 are modelled, so this option can be used with a suitable input field to select a subset of land points (e.g. those in a particular catchment). **FALSE: All points are modelled (land and sea).**
`subArea`	logical	Switch indicating if a subset of the input grid is to be selected. Only used if `pointsList`=FALSE. **TRUE: a subsection of the input grid will be used** (see `xcoord` and `ycoord` below) **FALSE: the full input grid is considered.**
`subAreaLatLon`	logical	If `subArea=TRUE`, this indicates how to interpret the coordinates `xcoord` and `ycoord`. **TRUE: co-ordinates are longitude and latitude.** **FALSE: co-ordinates are x and y indices (column and row numbers).**
`xcoord(1:2)`	real array	x-coordinates of the sub-area to be considered. Depending on `subAreaLatLon`, these are longitudes (in range -180 to 360°) or column numbers. See notes on grid definition in Section 6.4. If values are

		column numbers, the code uses the nearest integer to the input value.
`ycoord(1:2)`	real array	As `xcoord`, expect in latitudinal (y) direction.
`npoints`	integer	The number of points that are to be modelled. Only used if `pointsList`=TRUE.
`readFilePoints`	logical	Switch controlling source of list of point numbers. Only used if `pointsList=TRUE`. **TRUE: read from an external ASCII file** **FALSE: read from the run control file**. Points are specified at the sub-section marked `>DATA_POINTS` (see Section 6.4.3).
`fileNamePoints`	character	Name of file containing list of points. Only used if `pointsList=TRUE`.

6.4.2. `INIT_LAND`: Land fraction

This section describes how the land fraction field is set. Originally land fraction described the fraction of each gridbox that is land, but (offline) JULES can use the “land fraction” field as a mask that allows a subset of points to be modelled - e.g. “land fraction” can be set to 1 at all locations within a catchment, and to zero (or less) at all other points (such as land points outside the catchment). For this latter use, `landOnly` should be TRUE.

```
>INIT_LAND

readFileLand
fileFormatLand
fileNameLand

>ASCBIN
nheaderFileLand,nheaderFieldLand
fieldLand

>NC
varNameLand
```

Table 11 Description of variables in the INIT_LAND section

Variable name	Type and permitted values	Notes
`readFileLand`	logical	Switch controlling source of land fraction data. **TRUE: read from an external file** **FALSE: read from the run control file**, at the section marked `>DATA_LAND` (see Section 6.4.3).
`fileFormatLand`	character See Section 5.2.	Format of file containing land fraction data.

`fileNameLand`	character	Name of file containing land fraction data.
The following are read only if `readFileLand=TRUE`. Only values for the appropriate file format are read.		
>ASCBIN: If `fileFormatLand='asc','bin' or 'pp'`:		
`nheaderFile`	integer >=0	The number of headers at the start of the land fraction file. See Section 5.2.
`nheaderField`	integer >=0	The number of headers before each field in the land fraction file. See Section 5.2.
`fieldLand`	Integer >=1	The field number in the file that holds data for the first level of this variable. See discussion of fields in Section 5.1.
>NC: If `fileFormatLand='nc'`:		
`varNameLand`	character array	The name of the variable containing the land fraction.

6.4.3. INIT_LATLON: Latitude and longitude

```
>INIT_LATLON

regLatLon
regLat1, regLon1
regDlat, regDlon
readFileLatLon
fileFormatLatLon
fileNameLatLon

>ASCBIN
nheaderFile, nheaderField
fieldLat, fieldLon

>NC
varNameLat,varNameLon

>DATA_POINTS
pointList(1:npoints)
>DATA_LAND
flandg(1:nxIn,1:nyIn)
>DATA_LATLON
latitude(1:nxIn,1:nyIn)
longitude(1:nxIn,1:nyIn)
```

Table 12 Description of variables in the INIT_LATLON section.

Variable name	Type and permitted values	Notes

regLatLon	logical	Switch indicating if the input grid is 'regular' (and will be described by origin and increment) or if latitude and longitude fields are to be read. **TRUE: the grid is 'regular' and can be specified by its origin and gridbox size**. There is then no need to read lat/lon values for each gridpoint. **FALSE: read latitude and longitude values for each gridpoint.**
regLat1	real	The latitude (decimal degrees North) of the southernmost row of gridpoints in the input grid (NOT necessarily the model grid). The gridpoint is in the centre of the gridbox.
regLon1	real -180 to 360	The longitude (decimal degrees East) of the westernmost column of gridpoints in the input grid (NOT necessarily the model grid).
regDlat	real >0.0	The size of a gridbox in the NS direction (decimal degrees of latitude). Note: regLat1 and regLon1 are only used if regLatLon=TRUE. regDlat and regDlon may be used even if regLatLon=FALSE, if there is a need to establish the area of each gridbox (which is needed by some parameterisations and to label output).
regDlon	real >0.0	The size of a gridbox in the EW direction (decimal degrees of longitude).
readFileLatLon	logical	Switch controlling source of latitude and longitude data. Only used if pointsList=FALSE and regLatLon=FALSE. **TRUE: read from an external file** **FALSE: read from the run control file, at the section marked >DATA_LATLON.**
fileFormatLatLon	character	Format of file containing latitude and longitude data.
fileNameLatLon	character	Name of file containing latitude and longitude data.
The following are read only if readFileLatLon=TRUE. Only values for the appropriate file format are read.		
>ASCBIN: If fileFormatLatLon='asc', 'bin' or 'pp':		
nheaderFile	integer >=0	The number of headers at the start of the lat/lon file. See Section 5.2.
nheaderField	integer >=0	The number of headers before each field in the lat/lon file. See Section 5.2.
fieldLat	integer >=1	The field number in the file that holds latitude data. See discussion of fields in Section 5.1.
fieldLon	integer >=1	The field number in the file that holds longitude data.
>NC: If fileFormatLatLon='nc':		
varNameLat	character	The name of the variable containing the latitude data.
varNameLon	character	The name of the variable containing the longitude data.
The following sections are used only if the switches above indicate that the fields are to be read		

from the run control file.		
`pointList(1:npoints)`	integer array >=1	A list of the points that are to be modelled. These are point numbers in the input grid. NB If the input data run from North to South (i.e. not the JULES S to N order), the point numbers should still be calculated following the JULES S to N convention. Thus point number 1 is in the SW corner of the grid, which will not be the first point in the input data if `yrevIn`=T (unless `nyIn`=1).
`flandg(1:nxIn,1:nyIn)`	real array	The fraction of each gridbox that is land. If `landOnly`=TRUE, only locations with flandg>0 are modelled.
`Latitude(1:nxIn,1:nyIn)`	real array	The latitude of each gridpoint.
`Longitude(1:nxIn,1:nyIn)`	real array -180 to 360	The longitude of each gridpoint. All values should be in the range of either -180 to 180º or 0 to 360º.

The special case of an equal angle grid (all gridboxes have same extent in terms of latitude and longitude) in which the rows run WE and the columns SN (hereafter referred to as an equal angle grid), can be set up via a simple option. All other grids, including a vector of points, require the latitude and longitude of all points to be input.

If `regLatLon`=TRUE, the input data must be presented in the default JULES order (starting bottom left at (`regLat1,regLon1`) and proceeding row-wise). If `regLatLon`=FALSE, the input data need not be in order of lat/lon coordinates – each point in the grid will use the lat/lon read in for that point.

6.4.4. Examples of grid description

The latitude and longitude of the grid must be specified for all runs. For many model runs, the location of the grid is important, since it controls important factors such as the angle of the sun. Other, more idealised, runs might not need this information, but in this case the location may still be required so that the model output can be correctly mapped. If the location is not needed for either purpose, the user should enter an arbitrary location (e.g., 0°N, 0°E).

Grid example 1: A single point run.

This covers the simplest case: the input contains a single point. We assume that `nxIn=1` and `nyIn` =1 (see Section 6.2). All values are obtained from the run control file – no other file is involved. Only the lines in **bold** are relevant, and irrelevant sections have been omitted.

```
>INIT_GRID

T,F,F                 !   pointsList,coord,coordLL
F                     !   landOnly
F,F                   !   subArea, subAreaLatLon
1,2,3,4               !   xcoord(1:2),ycoord(1:2)
```

```
1                 !  npoints
F                 !  readFilePoints
'points.txt'      !  fileNamePoints

>INIT_LAND
F                 !  readFileLand
'bin'             !  fileFormatLand
'grid.gra'        !  fileNameLand

>INIT_LATLON
T                 !  regLatLon
40.0, 50.0        !  regLat1, regLon1
1.0,1.0           !  regDlat, regDlon
F                 !  readFileLatLon
'bin'             !  fileFormatLatLon
'latlon.gra'       !  fileNameLatLon

>DATA_POINTS
1                  !  pointList

>DATA_LAND
1.0                !  flandg

>DATA_LATLON
0.0                !  latitude
5.0                !  longitude
```

`pointsList=T` indicates that the grid will be described by a list of points.
`npoints=1` shows that this run is for a single point.
`readFilePoints=F` indicates that the point numbers are read from the `>DATA_POINTS` section, where point number 1 is indicated (the only possibility for an input grid of one point).
`readFileLand=F` indicates that the land fraction field is read from the `>DATA_LAND` section, where the value 1.0 shows that the single gridbox is 100% land.
`regLatLon=T` indicates that the grid is 'regular' and will be described by its origin (`regLat1`, `regLon1`) and gridbox size (`regDlat`, `regDlon`). There is then no need for any further information about coordinates – in particular the data at `>DATA_LATLON` are not read.

Grid example 2: Selecting points in a given range of latitude and longitude.

The grids used in this example are shown in Figure 1. The input grid has `nxIn=5`, `nyIn=4`, and we wish to model the area 55-57ºN 355-357ºE (5ºW-3ºW). To do this, we use the following entries in the run control file. Only the lines in **bold** are relevant, and irrelevant sections have been omitted.

```
>INIT_GRID

F,F,F                  !  pointsList,coord,coordLL
F                      !  landOnly
T,T                    !  subArea, subAreaLatLon
```

```
355.0,357.0,55.0,57.0  !  xcoord(1:2),ycoord(1:2)
1                      !  npoints
F                      !  readFilePoints
'points.dat'           !  fileNamePoints

>INIT_LAND
T                 !  readFileLand
'bin'             !  fileFormatLand
'grid.gra'        !  fileNameLand

>ASCBIN
0,0               !  nheaderFileLand,nheaderFieldLand
1                 !  fieldLand

>INIT_LATLON
T                 !  regLatLon
55.5, 353.5       !  regLat1, regLon1
1.0, 1.0          !  regDlat, regDlon
F                 !  readFileLatLon
'bin'             !  fileFormatLatLon
'grid.gra'        !  fileNameLatLon
```

pointsList=F indicates that the model grid will be determined by the land fraction field (and also latitude and longitude in this case).
landOnly=F indicates that both sea and land points will be selected.
subArea=T indicates a sub-section of the input grid is requested. subAreaLatLon=T indicates that the sub-section will be specified by a range of latitude and longitude, shown by xcoord and ycoord to be 355ºE to 357ºE,55·0ºN to 57·0ºN (note we could enter the longitude range as -5 to -3).
npoints is irrelevant because the number of points will be determined as the number of points the model finds within the given lat/lon range.
readFileLand=T indicates that the land fraction field is read from the binary file called 'grid.gra', which has no headers and contains land fraction as the first field.
regLatLon=T indicates that the input grid is a 'regular' grid, with origin (the gridpoint in the southwest corner) shown by regLat1, regLon1 to be 55.5ºN 355.5ºE, and gridbox size 1º×1º.

With this information, JULES determines that there are 4 gridpoints within the given lat/lon range, and that the model grid will be a square of side 2 gridboxes. The land fraction field shows that these are all land points, meaning that the land vector also has 4 points. Note that these points could also have been selected by providing a list of the point numbers, indicated by pointsList=TRUE, npoints=4, and then entering the point numbers (3, 4, 8, 9) after >DATA_POINTS.

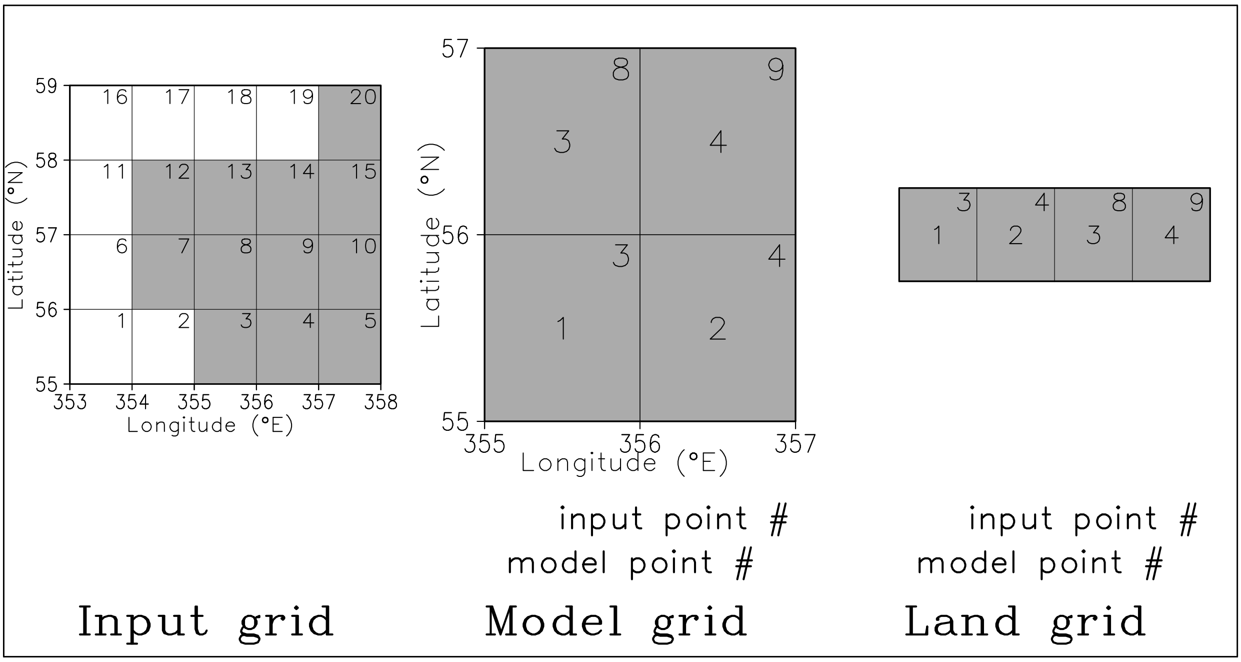

Figure 1. Example of grid selection based on longitude and latitude.

Grid Example 3: Selecting only land points in a given range of latitude and longitude.

This example is similar to Example 2, but this time we only wish to model land points within a given area. The grids used in this example are shown in Figure 2 and we wish to model land points in 55-57ºN 354-356ºE (6ºW-4ºW).

To do this, we use the following entries in the run control file. Only the lines in **bold** are relevant, and irrelevant sections have been omitted.

```
>INIT_GRID

F,F,F                  !  pointsList,coord,coordLL
T                      !  landOnly
T,T                    !  subArea, subAreaLatLon
-6.0,-4.0,55.0,57.0    !  xcoord(1:2),ycoord(1:2)
1                      !  npoints
F                      !  readFilePoints
'points.dat'           !  fileNamePoints

>INIT_LAND
T                  !  readFileLand
`bin'              !  fileFormatLand
`grid.gra'         !  fileNameLand
```

`pointsList`=F indicates that the model grid will be determined by the land fraction field (and also latitude and longitude in this case).
`landOnly`=T indicates that only land points will be selected.
`subArea`=T indicates a sub-section of the input grid is requested. `subAreaLatLon`=T indicates that the sub-section will be specified by a range of latitude and longitude, shown by `xcoord` and `ycoord` to be 6ºW to 4ºE, 55ºN to 57ºN.
`npoints` is irrelevant because the number of points will be determined as the number of land points the model finds within the given lat/lon range.
`readFileLand=T` indicates that the land fraction field is read from the binary file called 'grid.gra', which has no headers and contains land fraction as the first field.

With this information, JULES determines that there are 4 gridpoints within the given lat/lon range, but only 3 are land. As the 3 land points do not form a rectangle, the model grid is a vector of 3 points. As we are only modelling land points, the land grid is identical to the model grid.

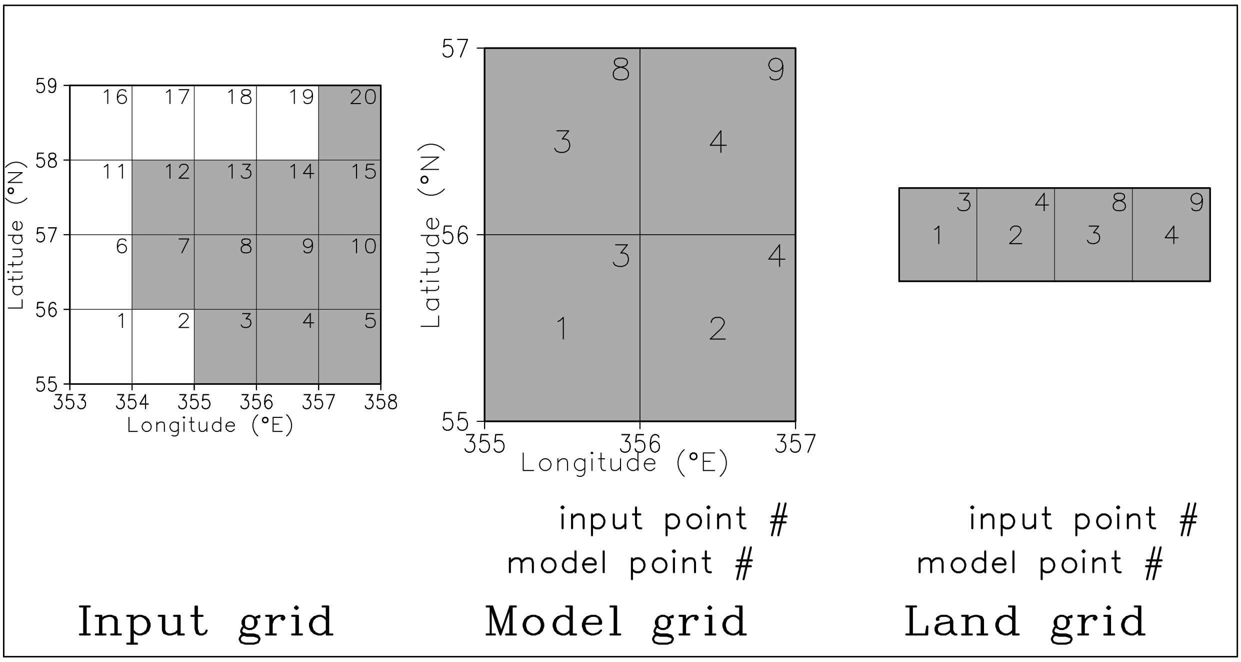

Figure 2 Example of grid selection based on longitude and latitude, taking land points only.

6.5. `INIT_FRAC`: Fractional coverage of land surface types

In this section, we specify the fraction of the land area in each gridbox that is covered by each of the surface types. Under certain circumstances (described below), this information may be acquired later, via another section.

```
>INIT_FRAC

readFracIC
readFile
fileFormat
filename

>ASCBIN
nheaderFile, nheaderField
fieldNum

>NC
varName

>DATA
frac(1:nxIn,1:nyIn)
```

Table 13 Description of variables in the `INIT_FRAC` section.

Variable name	Type and permitted values	Notes
`readFracIC`	logical	Switch indicating location of fractional cover data. **TRUE: fractional cover is provided as part of the initial condition in section INIT_IC** (see Section 6.20) **and is not provided here.** **FALSE: fractional cover will be read from this section.** For runs with dynamic vegetation (`l_veg_compete=TRUE`), the fraction cover is a prognostic variable and it must be read with the initial condition (`readFracIC=TRUE`).
`readFile`	logical	Switch controlling location of fractional cover data. Only used if `readFracIC=FALSE`. **TRUE: read from an external file** **FALSE: read from the run control file.**
`fileFormat`	character See notes in Section 5.2.	Format of data. Only used if `readFile=TRUE`.
`filename`	character	Name of file containing data. Only used if

		readFile=TRUE.
The following are read only if readFile=TRUE . Only values for the appropriate file format are read.		
>ASCBIN: The following are used if fileFormat='asc', 'bin' or 'pp'.		
nheaderFile	integer >=0	The number of headers at the start of the file. See Section 5.2.
nheaderField	integer >=0	The number of headers before each field. See Section 5.2.
fieldNum	integer >=1	The number of the first field to be used from the input file (this represents the first surface type). See discussion of fields in Section 5.1.
>NC: The following are used if fileFormat='nc'.		
varName	character	The name of the variable containing data.
>DATA: The following are used if readFile=FALSE.		
frac(1:nxIn,1:nyIn,1:ntype)	real array >=0.0	The fractional coverage of each surface type. The fractions should sum to 1 (this is checked by the code). These values are only read if readFile=F, and must be located after the tag >DATA. NB: If using either URBAN-2T or MORUSES then the **total** urban fraction should be entered in the 'urban_canyon' tile[5,6].

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.

6.5.1. Example: Reading frac from the run control file.

We assume nxIn=nyIn=npoints=1, and ntype=9. Only the lines relevant to this case are shown.

```
>INIT_FRAC

F             !   readFracIC
F             !   readFile

>DATA
0.55, 0.15, 0.20, 0.00, 0.05, 0.00, 0.05, 0.00  ! frac(1,1,1:ntype)
```

readFracIC=F indicates that frac is read here, rather than as part of the initial condition.
readFile=F indicates that data will be read from the run control file, not from another file.
The 9 values of frac are positioned after the >DATA tag.

[6] The total urban fraction only should be entered because the canyon and roof fractions are calculated using the canyon fraction (W/R). The canyon fraction is set in INIT_URBAN (Section 6.13) and can either be prescribed by the user or calculated by an empirical formula described in Table 25 under l_urban_empirical.

6.5.2. Example: Setting the same tile fractions on all land points

If we have more than one point on the input grid and 9 defined surface types (`npoints`>1, `ntype`=9), then it is possible to set the same fractions over all gridboxes without having to make separate input files that would contain the same information repeated `npoints` times. In this case with, say, `npoints`=1000, the relevant lines in the run control file are,

```
>INIT_FRAC

F             !   readFracIC
F             !   readFile

>DATA
1000*0.55
1000*0.15
1000*0.20
1000*0.00
1000*0.05
1000*0.00
1000*0.00
1000*0.05
1000*0.00     !  frac
```

It is significant that the data for each JULES surface type are written on a separate line, in contrast to the single grid point case where all values are written on one line and separated by commas. This is because these `frac` data are read one type at a time in blocks of all grid points (unless the input grid is a single point to be read from the run control file).

6.6. `INIT_SOIL`: Soil layer depths and hydraulic and thermal characteristics

In this section we specify the depth of each soil layer and also the hydraulic and thermal characteristics of the soil.

```
>INIT_SOIL

l_vg_soil
l_soil_sat_down
l_q10
soilhc_method

useSoilType
constZ,zrev
readFile
fileFormat
fileName
LUTfileName

>ASCBIN
nheaderFile,nheaderField
>VARS
name(1) fieldNumber(1)
---- Repeated for each variable. ---
>ENDVARS

>NC
>VARS
name(1),SDFname(1)
---- Repeated for each variable. ---
>VARS

>DATA_DZSOIL
dzsoil(1:sm_levels)
albSoilConst

>DATA
data values
```

Table 14 Description of variables in the INIT_SOIL section.

Variable name	Type and permitted values	Notes
`l_vg_soil`	logical	Switch for van Genuchten soil hydraulic model.

		TRUE: use van Genuchten model. **FALSE: use Brooks and Corey model.**[7] 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.
`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.
`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.**
`soilhc_method`	Integer Allowable values: 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).**
`useSoilType`	logical	Switch controlling how soil characteristics are input. TRUE: a map of soil types (classes) will be provided, along with a look-up table (LUT) giving the soil characteristics for each soil type. Each gridbox contains a single soil type, but the soil properties of that type can vary with depth. FALSE: maps of soil properties are provided.
`constZ`	logical	Switch indicating if soil characteristics are to be uniform with depth at each gridbox. Not used if useSoilType=TRUE. **TRUE: soil characteristics do not vary with depth.** **FALSE: soil characteristics vary with depth.**
`zrev`	logical	Switch indicating if input data are stored in reverse order of levels compared with JULES's expectation. **TRUE: vertical order is reversed, with data stored in**

[7] 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.

		"bottom to top" order (i.e. bottom layer first). **FALSE: standard vertical order, with data stored in "top to bottom" order (i.e. uppermost layer first).** Must be FALSE if useSoilType=TRUE.
`readFile`	logical	Switch controlling location of soil data. **TRUE: read from an external file** **FALSE: read from the run control file.**
`fileFormat`	character	Format of data file. Only used if `readFile=TRUE`.
`fileName`	character	Name of file containing data. Only used if `readFile=TRUE`.
`LUTfileName`	character	Name of file containing the look-up table (LUT) of soil characteristics for each soil type. Only used if `useSoilType`=TRUE. This is an ASCII file, the format of which is described in Section 0.
`>ASCBIN`: The following are used if `fileFormat='asc', 'bin' or 'pp'`, or if `readFile=FALSE`.		
`nheaderFile`	integer >=0	The number of headers at the start of the file (not used if `readFile`=FALSE). See Section 5.2.
`nheaderField`	integer >=0	The number of headers before each field (not used if `readFile`=FALSE). See Section 5.2.
Each variable is described by a line with two values (`name` and `fieldNumber`), separated by spaces (NB no commas). The list of variables is preceded by the tag `>VARS`, and followed by the tag `>ENDVARS`.		
`Name`	character	The name of a soil variable. These names must be chosen from the list in Table 15 List of soil parameters. below. If useSoilType=TRUE, only the soil type should be provided, otherwise all 9 other variables must be provided.
`fieldNumber`	integer	The field number of the first level of data in the input file that is to be used for a variable. See discussion of fields in Section 5.1. (Note that if `readFile=FALSE`, this is interpreted slightly differently – it is the variable number, not field number.)
`>NC`: The following are used if `fileFormat='nc'`.		
`name`	character	See under `>ASCBIN` above.
`SDFname`	character	The name of a variable containing data, as it appears in the SDF.
>DATA_DZSOIL		
`dzSoil(1:sm_levels)`	real array	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 its standard setup, JULES uses layer depths of 0.1, 0.25, 0.65 and 2.0m, giving a total depth of 3.0m.
`albSoilConst`	Real	A value of soil albedo that is to be used at all locations. Only used if `useSoilType`=TRUE.
`>DATA`: If `readFile=FALSE`, data for the soil variables should be listed here in the order given in Table 15 List of soil parameters. .		

Table 15 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
soilType	The soil type (class). Although this is an integer variable, it is treated as a real variable for convenience during input and output.

Names must be entered exactly as specified here (including case).
If useSoilType=FALSE, all variables other than soilType are required.
If useSoilType=TRUE, only soilType is required.

6.6.1. The soil look-up table file

The soil look-up table should be formatted as shown below, with the meaning of the variables described in Table 16.

```
### Header lines (e.g. containing a description of data sources)
### that are not to be read by JULES should begin with # or !.
nz
dzSoilLUT(1:nz)
nsoil

soilNum
soilChar(isoil,1:8,iz=1)
…
soilChar(isoil,1:8,iz=nz)
---- Repeated for each soil type ----
```

Table 16 List of variables in soil look-up table.

Variable name	Type and permitted values	Notes
nz	integer Must equal `sm_levels`.	Only used to check that LUT is consistent with current soil configuration.
dzSoil(1:nz)	integer array Must equal `dzSoil`.	Only used to check that LUT is consistent with current soil configuration.
nsoil	integer	The number of soil types in the file. Not all of these need to be present in the map of soil types. The
soilNum	integer	The soil number (a class or ID). These need not be consecutive. This number is used to map each value of `soilType` found in the map of soil types to a set of characteristics.
soilChar(1:nsoil,1:8,1:sm_levels)	real array	The soil characteristics for each soil type and each layer. NB Values are required for each layer, that is, a soil type implies a profile of values. The 8 characteristics must be given in the following order (see Table 15 List of soil parameters. for explanation of names): sathh, b, hcap, hcon, satcon, sm_crit,sm_sat,sm_wilt.

6.7. INIT_TOP: parameters for TOPMODEL

This section reads parameter values for the TOPMODEL-type parameterisation of runoff. It is only read if l_top=TRUE. The description below is very brief. For further details references under `l_top` in Section 6.2.

```
>INIT_TOP

zw_max
ti_max
ti_wetl

readFile
fileFormat
fileName
```

```
>ASCBIN
nheaderFile,nheaderField
>VARS
name(1) fieldNumber(1)
---- Repeated for each variable. ---
>ENDVARS

>NC
>VARS
name(1),SDFname(1)
---- Repeated for each variable. ---
>VARS

>DATA
data values
```

Table 17 Description of variables in the INIT_TOP section

`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.
`ti_max`	real	The maximum possible value of the topographic index. A value of 10 has been used successfully.
`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.
`readFile`	logical	Switch controlling location of soil data. **TRUE: read from an external file** **FALSE: read from the run control file.**
`fileFormat`	character	Format of data file. Only used if readFile=TRUE.
`fileName`	character	Name of file containing data. Only used if readFile=TRUE.
`>ASCBIN`: The following are used if `fileFormat='asc'`, `'bin'` or `'pp'`, or if `readFile=FALSE`.		
`nheaderFile`	integer >=0	The number of headers at the start of the file (not used if `readFile`=FALSE). See Section 5.2.
`nheaderField`	integer >=0	The number of headers before each field (not used if `readFile`=FALSE). See Section 5.2.
Each variable is described by a line with two values (`name` and `fieldNumber`), separated by spaces (NB no commas). The list of variables is preceded by the tag `>VARS`, and followed by the tag `>ENDVARS`.		
name	character	The name of a variable. These names must be chosen from

		the list in Table 18 below.
flag	Integer -1 or >0	Flag indicating how this variable should be set. -1: the following value of constVal will be used to set the value at all locations >0: The field number of the first level of data in the input file that is to be used for a variable. See discussion of fields in Section 5.1. (Note that if `readFile=FALSE`, this is interpreted slightly differently – it is the variable number, not field number.)
constVal	real	A value that is used to set a spatially constant field. Only used if flag=-1.
`>NC`: The following are used if `fileFormat='nc'`.		
name	character	See under `>ASCBIN` above.
SDFname	character	The name of a variable containing data, as it appears in the SDF.
flag	integer	Flag indicating how this variable should be set. -1: the following value of constVal will be used to set the value at all locations All other values are ignored and data from the SDF are used.
constVal	real	See under `>ASCBIN` above.
`>DATA`: If `readFile=FALSE`, data for the TOPMODEL variables should be listed here in the order given in Table 18.		

Table 18 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.

* Names must be entered exactly as specified here (including case).

6.8. INIT_PDM: parameters for PDM

This section reads parameter values for the PDM-type parameterisation of surface runoff. It is only read if `l_pdm`=TRUE. Note that these parameters are held constant across the model domain. For further details of PDM, see references under `l_pdm` in Section 6.2.

```
>INIT_PDM

dz_pdm
b_pdm
```

Table 19 Description of variables in the INIT_PDM section.

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

6.9. INIT_HGT: elevation of tiles

This section 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.

```
>INIT_HG

zeroHeight

readFile
fileFormat
fileName

>ASCBIN
nheaderFile,nheaderField
fieldNum

>NC
SDFname

>DATA
data values
```

Table 20 Description of variables in the INIT_HGT section

zeroHeight	logical	Switch used to simplify the initialisation of tile elevation. **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 to follow.
readFile	logical	Switch controlling location of elevation data. **TRUE: read from an external file** **FALSE: read from the run control file.**
fileFormat	character	Format of data file. Only used if readFile=TRUE.
fileName	character	Name of file containing data. Only used if readFile=TRUE.
>ASCBIN: The following are used if fileFormat='asc', 'bin' or 'pp', or if readFile=FALSE.		

nheaderFile	integer >=0	The number of headers at the start of the file (not used if readFile=FALSE). See Section 5.2.
nheaderField	integer >=0	The number of headers before each field (not used if readFile=FALSE). See Section 5.2.
fieldNum	integer >=1	The number of the first field to be used from the input file (this represents the first surface tile). See discussion of fields in Section 5.1.
>NC: The following are used if fileFormat='nc'.		
SDFname	character	The name of a variable containing data, as it appears in the SDF.
>DATA: If readFile=FALSE, data for the tile elevations variables should be listed here. Values for each tile should be listed on a separate line.		

6.10. INIT_VEG_PFT: Time-invariant parameters for plant functional types

This section 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/or location are dealt with in control file section INIT_VEG_VARY (see guide Section 6.11). Parameters that are only required if the dynamic vegetation (TRIFFID) or phenology sections are requested are read separately in control file section INIT_TRIF (see guide Section 6.15).

For many applications, the best approach may be to read the PFT parameters from the standard parameter files provided with the JULES code (readFile=TRUE, filename='PARAM/standard_pft_param.dat'), since this removes the risk that values can be changed by an accidental edit to the run control file. The description of INIT_VEG_PFT options is given in Table 21 and the list of required variables is given in Section 6.10.

```
>INIT_VEG_PFT

readFile
fileName
npftInFile

>DATA
var1(1),var1(2),...,var1(npft)
var2(1),var2(2),...,var2(npft)
… … data values … …
```

Table 21 Description of variables in the INIT_VEG_PFT section.

Variable name	Type and permitted values	Notes
readFile	logical	Switch controlling location of data. TRUE: read from an external file

		FALSE: read from the run control file.
filename	character	The name of the external file containing the data. Only used if readFile=TRUE.
npftInFile	integer ≥ npft	The number of PFTs for which parameters are given in the input file.
>DATA: If readFile=FALSE, the data should be listed here (on the line after >DATA) in the order given in Section 6.2. Each variable should start on a new line, and npftInFile values should be given.		

Each parameter has a separate value for each PFT, npftInFile values are read for each parameter. All values are of type REAL, unless stated otherwise. Parameters for the TRIFFID or phenology modules are described in Section 6.15.
HCTN24 and 30 refer to Hadley Centre technical notes 24 and 30, available from http://www.metoffice.gov.uk/publications/HCTN

Table 22 List of PFT parameters.

Variable name	Description
typeName	Character. Name of each PFT. This list must include the PFTs used in this run – see pftName in section INIT_OPTS (Section 6.2). These names are for the user's convenience, and do not have any special significance within JULES.
c3	Integer. Flag indicating whether PFT is C3 type. 0 : not C3 (i.e. C4) 1 : C3
canht_ft	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_trif=FALSE). If TRIFFID is active, canht_ft is a prognostic variable and its initial value is read as described in Section 6.20 below.
LAI	The leaf area index (LAI) of each PFT. The value read here is only used if TRIFFID is not active (l_trif=FALSE). If TRIFFID is active, LAI is a prognostic variable and its initial value is read as described in Section 6.20 below.
catch0	Minimum canopy capacity (kg m^{-2}). This is the minimum amount of water that can be held on the canopy. See HCTN30 p7.
dcatch_dlai	Rate of change of canopy capacity with LAI (kg m^{-2}). Canopy capacity is calculated as catch0 + dcatch_dlai*lai. See HCTN30 p7.
dz0v_dh	Rate of change of vegetation roughness length for momentum with height. Roughness length is calculated as dz0v_dh*canht_ft. See HCTN30 p5.
Z0h_z0m	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).
infil_f	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.
rootd_ft	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.
`snowCanPFT`	Flag indicating whether snow can be held under the canopy of each PFT. Only used if `can_model=4` (see Section 6.2). The model of snow under the canopy is currently only suitable for coniferous trees. Acceptable values are: 0: snow cannot be held under the canopy. 1: snow can be held under the canopy.
`albsnc_max`	Snow-covered albedo for large leaf area index. Only used if `l_spec_albedo=FALSE`. See HCTN30 Eq.2
`albsnc_min`	Snow-covered albedo for zero leaf area index. Only used if `l_spec_albedo=FALSE`. See HCTN30 Eq.2.
`albsnf_max`	Snow-free albedo for large LAI. Only used `if l_spec_albedo=FALSE`. See HCTN30 Eq.1.
`kext`	Light extinction coefficient - used with Beer's Law for light absorption through tile canopies. See HCTN30 Eq.3.
`kpar`	PAR Extinction coefficient (m^2 leaf/m^2 ground)
`orient`	Flag indicating leaf angle distribution. 0 : spherical 1 : horizontal
`alpha`	Quantum efficiency (mol CO_2 per mol PAR photons).
`alnir`	Leaf reflection coefficient for NIR. HCTN30 Table 3
`alpar`	Leaf reflection coefficient for VIS. HCTN30 Table 3
`omega`	Leaf scattering coefficient for PAR.
`omnir`	Leaf scattering coefficient for NIR.
`a_wl`	Allometric coefficient relating the target woody biomass to the leaf area index (kg carbon m^{-2}).
`a_ws`	Woody biomass as a multiple of live stem biomass.
`b_wl`	Allometric exponent relating the target woody biomass to the leaf area index. This is 5/3 in HCTN24 Eq.8.
`eta_sl`	Live stemwood coefficient (kg C/m/LAI)
`g_leaf_0`	Minimum turnover rate for leaves (/360days).
`dgl_dm`	Rate of change of leaf turnover rate with moisture availability.
`dgl_dt`	Rate of change of leaf turnover rate with temperature (K^{-1}). This is 9 in HCTN24 Eq.10.
`glmin`	Minimum leaf conductance for H_2O (m s^{-1}).
`dqcrit`	Critical humidity deficit (kg H_2O / kg air). See Eqn.17 of Cox et al. (1999).
`fd`	Scale factor for dark respiration. See HCTN 24 Eq. 56.
`f0`	`CI/CA` for `DQ = 0`. See HCTN 24 Eq. 32.
`fsmc_of`	Moisture availability below which leaves are dropped.
`neff`	Scale factor relating V_{cmax} with leaf nitrogen concentration. See HCTN 24 Eq. 51.
`nl0`	Top leaf nitrogen concentration (kg N/kg C).
`nr_nl`	Ratio of root nitrogen concentration to leaf nitrogen concentration

ns_nl	Ratio of stem nitrogen concentration to leaf nitrogen concentration.
r_grow	Growth respiration fraction
sigl	Specific density of leaf carbon (kg C/m2 leaf).
tleaf_of	Temperature below which leaves are dropped (K).
Tlow	Lower temperature for photosynthesis (deg C).
Tupp	Upper temperature for photosynthesis (deg C).
emis_pft	Surface emissivity
fl_o3_ct	Critical flux of O3 to vegetation (nmol/m2/s)
dfp_dcuo	Fractional reduction of photosynthesis with the cumulative uptake of O3 by leaves (/mmol/m2)

6.11. `INIT_VEG_VARY`: Time-/space- varying parameters for plant functional types

This section describes prescribed characteristics of the vegetation that vary with time and/or location, in addition to varying with PFT.

```
>INIT_VEG_VARY

nvegVar
vegDataPer, vegUpdatePer
nvegFileTime, vegFilePer
vegClim
readList
fileName
vegFileDate(1),vegFileTime(1)
vegEndTime
fileFormat

>ASCBIN
nfieldFile
nheaderFile,nheaderField
noNewLineVeg
varName(1),flag(1),fieldNumber(1),interp(1),nameFile(1)
--- Repeated for each of nvegVar variables.---

>NC
varName(1),flag(1),interp(1),SDFname(1),nameFile(1)
--- Repeated for each of nvegVar variables.---
```

Table 23 Description of variables in the INIT_VEG_VARY section.

Variable name	Type and permitted values	Notes
`nvegVar`	integer 0≤`nvegVar`≤ 3	The number of prescribed characteristics that vary with time and/or location. The three characteristics that may vary are vegetation height, leaf area index and root depth. If `nvegVar=0`, nothing more is read from this section.
`vegDataPer`	integer	The period (s) of time-varying data. If there are no time-varying fields, enter 0. Special cases: -1 indicates monthly data.
`vegUpdatePer`	integer	The period (s) between updates of time-varying fields. This must be less than or equal to the data period (`vegDataPer`). For example, `vegDataPer=86400, vegUpdatePer=3600`, indicates that the data are daily values and these

		should be updated (by interpolation) on an hourly basis. Special cases: 0: update every timestep -1: update once a month
`nvegFileTime`	integer ≥1	The number of data files available for each variable, each file holding data for different times. If all variables are held together, this is the number of data files. If variables are held in separate files, this is the number of files for any one variable.
`vegFilePer`	integer	The period (s) of the files containing the data. This must be at least as large as the period of the data (vegDataPer), and must be a multiple of the model timestep. Special cases: -1: monthly files -2: annual files
`vegClim`	logical	Switch indicating if time-varying vegetation data are to be treated as climatological, in the sense that the same data are to be used regardless of the year. TRUE: data are climatological. The year given for each file is ignored. FALSE: data are not climatological
`readList`	logical	Switch controlling how the names of the files containing the vegetation data, and the times covered by each, are read. TRUE: a list of names and times is read from another file. This is required if `nvegFileTime`>1. FALSE: a single name and file are read from the run control file. This option is only allowed if `nvegFileTime`=1 (see above).
`filename`		If `nvegFileTime=1` this is the name of the single data file (or the template). If `nvegFileTime>1`, this is the name of a file that lists the names and times of the data files. The first line of this file will be skipped (and so can be used for comments). All other lines are to be of the form `filename, startDate, "startTime"`, where `fileName` may contain variable-name-templating (see Section 6.22). `startDate` is in the format `yyyymmdd`, and time is in the format `hh:mm:ss`.
`vegFileDate`	integer	Date of first data in vegetation file, in format `yyyymmdd`. Only used if `readList=FALSE` (otherwise read from an external file).
`vegFileTime`	character	Time of first data in vegetation file, in format `hh:mm:ss`. Only used if `readList=FALSE` (otherwise read from an external file). **It is recommended that all times entered in JULES use Greenwich Mean Time (GMT or UTC), not**

		local time. The time zone used here must match that under INIT_TIME (see Section 6.3).
`vegEndTime`	logical	Flag used with vegetation file templating. `TRUE` means that time in filename refers to the final data in the file, `FALSE` means the time in the filename refers to the first data in the file.
`fileFormat`	character See Section 5.2.	Format of vegetation data files.
The following are read only if `readFile=TRUE`. Only values for the appropriate file format are read.		
>ASCBIN: If `fileFormat='asc', 'bin' or 'pp'`:		
`nfieldFile`	integer	Number of fields in each file.
`nHeaderFile`	integer >=0	The number of headers at the start of each file - see Section 5.2.
`nHeaderTime`	integer >=0	The number of headers at the start of each time - see Section 5.2.
`nHeaderField`	integer >=0	The number of headers at the start of each field - see Section 5.2.
`noNewLineVeg`	logical	Switch describing format of ASCII file. TRUE means that variables are arranged across one or more lines, and each variable does not necessarily start a new line. This option should be used if all the data for each time are one line of the input file (although it can also be used if the data continue onto subsequent lines). TRUE is only allowed if the fields are not functions of position (i.e. `vegFlag='t'`, see above). FALSE means that each variable starts on a new line.
`varName`	character `'canht'`, `'lai'`, `'rootd'`	The name of the variable. This is used to identify the variable in the code, and is set in the code. These must be entered exactly as listed, and are case-sensitive. Acceptable values: '`lai`' for leaf area index '`canht`' for canopy height '`rootd`' for root depth
`flag`	character `'t'`, `'tx'`, `'x'`	Flag indicating how the characteristic varies. Acceptable values: `t`: function of PFT and time only `tx`: function of PFT, time and location `x`: function of PFT and location only At present, all nvegVar variables must have the same value for this flag. `rootd` can only use flag '`t`' (i.e. root depth cannot vary with location in the current code).
`fieldNumber`	integer	The field number of the first level of data in the input file that is to be used for a variable.
`interpFlag`	character	Flag indicating how variable is to be interpolated in

	See Table 45.	time.
`nameFile`	character	The substitution string used in the names of files that contain this variable. Only used if variable name templating is used (see Section 6.22).
>NC: If `fileFormat='nc'`:		
`varName`	character	See above under `>ASCBIN`.
`flag`	character	See above under `>ASCBIN`.
`interpFlag`	character	See above under `>ASCBIN`.
`SDFname`	character	The name of the variable as it appears in a SDF.
`nameFile`	character	See above under `>ASCBIN`.

6.11.1. Examples of `INIT_VEG_VARY`

Example 1: Time-varying Leaf Area Index.

Leaf Area Index is to vary with time (but not with position on the grid). Climatological monthly data are to be used, with values updated at the start of each day. Note that the values are always assumed to be a function of PFT. The ASCII input file is illustrated in Figure 3 and contains one month of data (for all PFTs) on a single line.

```
Month p1  p2   p3   p4   p5
1     0.5  4.0  1.0  2.0  1.0
2     0.7  4.0  1.1  2.0  1.5
3     0.9  4.2  1.5  2.0  2.0
4     2.0  4.5  2.0  2.0  2.5
---- rest of file not shown---
```

Figure 3 Schematic of an ASCII file with monthly LAI data

The relevant entries in the run control file are shown below. Only the lines in **bold** are relevant, and irrelevant sections have been omitted.

```
>INIT_VEG_VARY

1                !  nvegVar
-1,86400         !  vegDataPer, vegUpdatePer
1,1              !  nvegFileTime, vegFilePer
T                !  vegClim
F                !  readList
'lai_monthly.dat'    !  fileName
20120115,'00:00:00'  !  vegFileDate(1),vegFileTime(1)
'asc'                !  fileFormat

>ASCBIN
6                !  nfieldFile
1,0              !  nheaderFile,nheaderField
```

```
T                  !  noNewLineVeg
'lai',    't', 2,  'i', 'notused'   ! name, flag, field, interp, nameFile
```

nvegVar=1 indicates that we only want to vary one vegetation characteristic. vegFileDate=20120115, but since vegClim=T, the year is discarded (effectively leaving 0115=15 January), meaning that each time of data is valid on the 15th of the month. nfieldFile=6 because we have data for each of 5 PFTs, plus there is a 'timestamp' variable that will not be used (see Figure 3). The final line shows that we want to vary LAI as a function of time (and PFT) only. The LAI data start with field #2. The 'I' and vegUpdatePer=86400 indicate that the monthly data will be interpolated between the monthly values and updated once every 86400s (once a day).

6.12. **INIT_NONVEG**: Parameters for non-vegetation surface types

```
>INIT_NONVEG

readFile
fileName
nnvgInFile

>DATA
dataVar1(1),dataVar1(2),...,dataVar1(nnvgInFile)
dataVar2(1),dataVar2(2),...,dataVar2(nnvgInFile)
… … data values … …
```

Table 24 Description of variables in the INIT_NONVEG section.

Variable name	Type and permitted values	Notes
`readFile`	logical	Switch controlling location of data. TRUE: read from an external file FALSE: read from the run control file.
`filename`	character	The name of the file to be read. Only used if readFile=TRUE. Note: For many applications, the best approach may be to read the parameters from the files provided with the JULES code (via `readFile=TRUE`), since this removes the risk that values can be changed by an accidental edit to the run control file.
`nnvgInFile`	integer ≥ nnvg	The number of non-vegetation surface types for which parameters are available in the input file.
`>DATA` The following is the list of `dataVar` parameters that must be defined for each non-PFT tile type. HCTN30 refers to Hadley Centre technical note 30, available from http://www.metoffice.gov.uk/publications/HCTN		
`typeName`	character	Name of each surface type. This list must include the non-vegetation surface types used in this run as defined in `INIT_OPTS` variable `nvgName` (see Section 6.2). Special cases: `'soil'` – this surface type must always be present. `'water'` – this is used to indicate open water, such as lakes. `'ice'` – this is used to indicate land ice, such as glaciers. `'urban_roof'` – this is used to indicate the urban roof tile. It enables the two-tile urban schemes and should be used in conjunction with `'urban_canyon'` (though see footnote 5 on page 28) `'urban_canyon'` – this is used to indicate the urban

		canyon tile. Must be used in conjunction with 'urban_roof' and **cannot** be used with 'urban' Each special type must be represented by not more than one type (e.g. we cannot have two 'soil' types).
albsnc_nvg	real	Snow-covered albedo. Only used if l_spec_albedo=FALSE. See HCTN30 Table 1
albsnf_nvg	real	Snow-free albedo. See HCTN30 Table 1 Only used if l_spec_albedo=FALSE.
catch_nvg	real	Capacity for water (kg m^{-2}). See HCTN30 p7
gs_nvg	real	Surface conductance (m s^{-1}). See HCTN30 p7 Soil conductance is modified by soil moisture according to HCTN30 Eq 35.
infil_nvg	real	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
z0_nvg	real	Roughness length for momentum (m). See HCTN30 Table 4
z0h_z0m	real	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.
ch_nvg	real	Heat capacity of this surface type (J K^{-1} m^{-2}). Used only if can_model is 3 or 4 (See INIT_OPTS, Section 6.2).
vf_nvg	real 0≤vf_nvg≤1	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 (See INIT_OPTS, Section 6.2)
emis_nvg	real	Surface emissivity.

6.13. **`INIT_URBAN`**: Urban model configuration, geometry & material characteristics

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 (though see footnote 5 on page 28). This section is only read if either of the two-tile urban schemes are enabled by including the `'urban_roof'` tile. The `'urban_roof'` and `'urban_canyon'` tile type parameters specified in `INIT_NONVEG` (Section 6.12) will be used for values that MORUSES does not parameterise, and for any MORUSES parametrisations that are turned off, according to Table 26. Further information on MORUSES, including references, can be found in the technical documentation and under `l_moruses` in `INIT_OPTS` (Section 6.2)

```
>INIT_URBAN

l_urban_empirical,l_moruses_macdonald
l_moruses_albedo,l_moruses_emissivity,l_moruses_rough
l_moruses_storage,l_moruses_storage_thin

anthrop_heat_scale

readFile
fileFormat
fileName

>ASCBIN
nheaderFile, nheaderField
>VARS
varName(1)   varFlag(1)   constVal(1)
varName(2)   varFlag(2)   constVal(2)
--- Repeat for each variable. ---
>ENDVARS

>NC
>VARS
varName(1)   varFlag(1)   constVal(1)   SDFname(1)
varName(2)   varFlag(2)   constVal(2)   SDFname(2)
--- Repeat for each variable. ---
>ENDVARS

# Data fields to be read from this file should appear below here.
>DATA
```

Table 25 Description of variables that are required in the INIT_URBAN section.

Variable name	Type and permitted values	Notes
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 27) 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)
The following are the parameterisation switches for the configuration of MORUSES. Where appropriate Table 26 gives the 'urban_roof' and 'urban_canyon' parameters that are required to be set in INIT_NONVEG (Section 6.12).		
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 27). 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
l_moruses_albedo	logical	MORUSES switch for effective canyon albedo parameterisation. The roof albedo is given by INIT_NONVEG (Section 6.12). TRUE: Use MORUSES parameterisation. Requires that **l_cosz = TRUE**, which the code automatically enables. FALSE See Table 26

l_moruses_emissivity	logical	MORUSES switch for effective canyon emissivity parameterisation. The roof emissivity is given by INIT_NONVEG (Section 6.12). TRUE: Use MORUSES parameterisation FALSE See Table 26
l_moruses_rough	logical	MORUSES switch for effective roughness length for heat parameterisation. TRUE: Use MORUSES parameterisation FALSE See Table 26
l_moruses_storage	logical	MORUSES switch for thermal inertia and coupling with underlying soil parameterisation TRUE: Use MORUSES parameterisation FALSE See Table 26
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
Other URBAN-2T and MORUSES options		
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
The following are give information about the source of data for urban geometry and building material properties		
readFile	logical	Switch that indicates source of data. TRUE: data are read from a named, external file FALSE: data are read from the run control file after the section marked >DATA
fileFormat	character	Flag indicating the file format. Case sensitive. Only used if readFile=.TRUE. 'asc': ASCII 'bin': generic binary (including GrADS) 'nc': netCDF 'pp': PP format

`fileName`		Name of file containing urban geometry & building material characteristics Only used if `readFile=.TRUE.`
`>ASCBIN`: The following are used if `fileFormat='asc'`, `'bin'`, or `'pp'`, or if `readFile = FALSE`.		
`nheaderFile`	integer	The number of headers at the start of the file. See Section 5.2
`nheaderField`	integer	The number of headers at the start of a field. See Section 5.2
`varName`	character	The name of the variable (see Table 27 Description of urban geometry & building material variables).
`varFlag`	integer ≥-1	Flag indicating how the variable is initialised. Acceptable values: >0: The field number in the file that holds data for this variable. See discussion of fields in Section 5.1. -1: The field will be set to the value `constVal` (see below) at all points. This option can be used to specify an idealised initial condition.
`constVal`	real	The value to be used at all points. Only used if `flag=-1`.
`>NC`: The following are used if `fileFormat='nc'`.		
`varName`	character	See under `>ASCBIN` above.
`varFlag`	integer ≥-1	Flag indicating how the variable is initialised. Acceptable values: >0: Default (effectively is ignored). -1: See under `>ASCBIN` above.
`constVal`	real	See under `>ASCBIN` above.
`SDFname`	character	The name of the variable as it appears in the SDF.
`>DATA` If `readFile = FALSE`, data should now appear in the run control file, in the order indicated by the value of `varFlag` for each variable (see `>ASCBIN` above) with each starting on a separate line.		

Table 26 Parameters that may be used from `INIT_NONVEG` (Section 6.12) 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 `INIT_NONVEG`.

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, z0h_z0m
l_moruses_storage	urban_canyon urban_roof	MORUSES	ch_nvg, vf_nvg

Table 27 Description of urban geometry & building material variables

Variable name	Description[8]	Notes on when data is not used. If not used / updated with calculated values then the variable could be set to constVal 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
emisw	Wall emissivity	Data only used if l_moruses_emissivity = TRUE
emisr	Road emissivity	See for emisw above

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

6.14. `INIT_SNOW`: Snow parameters

```
 >INIT_SNOW

dzSnow
rho_snow_const
snow_hcap,snow_hcon
snowLiqCap
r0,rmax
snow_ggr(1:3)
amax(1:2)
dtland,kland
maskd
snowLoadLAI,snowInterceptFact,snowUnloadFact
```

Table 28 Description of variables in the INIT_SNOW section

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

Variable name	Type and permitted values	Notes
`dzSnow(1:nsmax)`	array	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 29. `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.

rho_snow_const	real	Constant density of lying snow (kg m^{-3}). This is used if nsmax=0, or if the snowpack is very thin. It is also used as the density of fresh snow.
snow_hcap	real	Thermal capacity of lying snow (J K^{-1} m^{-3}) Typical value=0. ·3e6
snow_hcon	real	Thermal conductivity of lying snow (W m^{-1} K^{-1}) See HCTN30 Eq.42 Typical value= 0·265
snowLiqCap	real	Liquid water holding capacity of lying snow, as a fraction of snow mass. Only used if nsmax>0.
r0	real	Grain size for fresh snow (μm). See HCTN30 Eq.15. A typical value is 50·0. Only used if l_spec_albedo=TRUE.
rmax	real	Maximum snow grain size (μm). See HCTN30 p4. A typical value 2000.0 Only used if l_spec_albedo=TRUE.
snow_ggr(1:3)	real array	Snow grain area growth rates (μm ^{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. Typical values are 0·6, 0·06, 0·23e6
amax(1:2)	real array	Maximum albedo for fresh snow. . Only used if l_spec_albedo=TRUE. Values 1 and 2 are for VIS and NIR wavebands respectively. Typical values=0·98, 0. ·7
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.
kland	real	Used in snow-ageing effect on albedo. This is 0·3 in HCTN30 Eq4 (note the last term of that equation should be divided by dtland, i.e. kland as specified here includes a factor dtland in the denominator). Only used if l_spec_albedo=FALSE. Must not be zero.
maskd	real	Used in exponent of equation weighting snow-covered and snow-free albedo. This is 0·2 in HCTN30 Eq.5.
snowLoadLAI	real	Ratio of maximum canopy snow load to leaf area index (kg m^{-2}). This is 4·4 in JULES1. Only used if can_model=4.
snowInterceptFact	real	Constant in relationship between mass of intercepted snow and snowfall rate. This is 0·7 in JULES1. Only used if can_model=4
snowUnloadFact	real	Constant in relationship between canopy snow unloading and canopy snow melt rate. This is 0·4 in JULES1. Only

		used if `can_model`=4

Table 29 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 beign added at the same time. Don't panic - this should not be a problem for the simulation.

Table 29 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.

6.15. `INIT_TRIF`: Parameters for the TRIFFID model

This section is used to read PFT parameters that are only needed by the dynamic vegetation model (TRIFFID). Values are not read if TRIFFID is not selected. TRIFFID also uses many other PFT-specific variables that are also used in other parts of JULES, and are read in Section 6.10 above.

```
>INIT_TRIF

readFile
fileName
nnvgInFile

>DATA
dataVar1(1),dataVar1(2),…,dataVar1(nPft)
dataVar2(1),dataVar2(2),…,dataVar2(nPft)
… … data values … …
```

Table 30 Description of variables in the INIT_TRIF section.

Variable name	Type and permitted values	Notes
`readFile`	logical	Switch controlling location of data. TRUE: read from an external file FALSE: read from the run control file.
`filename`	`character`	The name of the file to be read. Only used if `readFile=TRUE`.
`npftInFile`	`integer` `≥npft`	The number of PFTs for which parameters are available in the input file.
>DATA If `readFile`=FALSE, the `dataVar` parameters should be listed in the order given below.		
`pftName`	character	Name of each PFT. These must match those given in Section 6.2.
`crop`	integer 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
`g_area`	`real`	Disturbance rate (/360days).
`g_grow`	`real`	Rate of leaf growth (/360days).
`g_root`	`real`	Turnover rate for root biomass (/360days).
`g_wood`	`real`	Turnover rate for woody biomass (/360days)
`lai_max`	`real`	Maximum LAI
`lai_min`	`real`	Minimum LAI

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

6.16. `INIT_AGRIC`: Fractional coverage by agriculture

If the TRIFFID vegetation model is used, the fractional area of agricultural land in each gridbox is read in this section. Otherwise, this section is not used.

```
>INIT_AGRIC

readFile
fileFormat
fileName

>ASCBIN
nheaderFile,nheaderField
fieldNum

>NC
varName

# Data fields to be read from this file should appear below here.
>DATA
frac_agr(1:nxIn,1:nyIn)
```

Table 31 Description of variables in the `INIT_AGRIC` section

Variable name	Type and permitted values	Notes
`readFile`	logical	Switch controlling location of soil layer data. TRUE: read from an external file FALSE: read from the run control file.
`fileFormat`	character	Format of data file. Only used if `readFile=TRUE`.
`filename`	character	Name of file containing data. Only used if `readFile=TRUE`.
The following are read only if `readFile=TRUE`. Only values for the appropriate file format are read.		
`>ASCBIN`: The following are used if `fileFormat='asc', 'bin' or 'pp'`.		
`nheaderFile`	integer >=0	The number of headers at the start of the file,
`nheaderField`	integer >=0	The number of headers before each field.
`fieldNum`	integer >=1	The field number of the first field to be used from the input file.
`>NC`: The following are used if `fileFormat='nc'`.		
`SDFName`	character	The name of the variable containing data, as it appears in the SDF.

>DATA: The following are used if readFile=FALSE.		
frac_agr(1:nxIn,1:nyIn)	real array	The fraction that is agriculture.

6.17. `INIT_MISC`: Miscellaneous surface, carbon and vegetation parameters

```
>INIT_MISC

hleaf,hwood
beta1,beta2
fwe_c3, fwe_c4
q10_leaf
kaps
kaps_roth(1:4)
q10_soil
cs_min
co2_mmr
frac_min, frac_seed
pow
```

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

Table 32 Description of variables in the INIT_MISC section

Variable name	Type and permitted values	Notes
`hleaf`	`real`	Specific heat capacity of leaves (J K^{-1} per kg carbon). HCTN30 p6 Typical value=5.7E4
`hwood`	`real`	Specific heat capacity of wood (J K^{-1} per kg carbon). HCTN30 p6 Typical value=1.1e4
`beta1`	`real`	Coupling coefficient for co-limitation in photosynthesis model. Cox et al. (1999), Eq.61 Typical value=0.83
`beta2`	`real`	Coupling coefficient for co-limitation in photosynthesis model. Cox et al. (1999), Eq.62 Typical value=0.93
`fwe_c3`	`real`	Constant in expression for limitation of photosynthesis by transport of products, for C3 plants. This is 0.5 in Eq.60 of Cox et al. (1999).
`fwe_c4`	`real`	Constant in expression for limitation of photosynthesis by transport of products, for C4 plants. This is 2.0×10^4 in Eq.60 of Cox et al. (1999).
`q10_leaf`	`real`	Q10 factor for plant respiration. Cox et al. (1999) Eq.66 Typical value=2.0
`kaps`	`real`	Specific soil respiration rate at 25 degC and optimum soil

		moisture (s^{-1}). Only used if not using TRIFFID (l_trif=FALSE). HCTN24 Eq.16. Typical value=5e-9
`Kaps_roth(1:4)`	`real`	Specific soil respiration rate for the RothC submodel for each soil carbon pool. Only used if using the TRIFFID vegetation model (l_trif=TRUE), in which case soil carbon is modelled using four pools (biomass, humus, decomposable plant material, resistant plant material).
`q10_soil`	`real`	Q10 factor for soil respiration. Only used if `l_q10`=TRUE. HCTN24 Eq.17 Typical value=2.0
cs_min	`real`	Minimum allowed soil carbon (kg m^{-2}) Typical value=1.0e-6
co2_mmr	`real`	Concentration of atmospheric CO2, expressed as a mass mixing ratio.
`frac_min`	`real`	Minimum fraction that a PFT is allowed to cover if TRIFFID is used. Typical value=1.0e-6
`frac_seed`	`real`	Seed fraction for TRIFFID. Typical value=0.01
pow	`real`	Power in sigmodial function used to get competition coefficients. This is 20.0 in HCTN24 Eq.3.

6.18. INIT_IMOGEN: Settings for the IMOGEN impacts model

```
>INIT_IMOGEN

imogenOrderFile
imogenNamelistFile
```

Table 33 Description of variables in the INIT_IMOGEN section

Variable name	Type and permitted values	Notes
imogenOrderFile	character	Path to file containing the mapping of IMOGEN global grid points onto IMOGEN land points (different from the JULES land points).
imogenNamelistFile	character	Path to file containing the IMOGEN namelists (described below)

IMOGEN uses Fortran namelists to set it's variables. These namelists are read from imogenNamelistFile . Below is an example of the IMOGEN namelist file - every possible value in each namelist is shown below along with it's default value. However the user need not specify every variable – those not specified will take their default value.

```
&ANLG_VALS
 Q2CO2=3.74
 F_OCEAN=0.711
 KAPPA_O=383.8
 LAMBDA_L=0.52
 LAMBDA_O=1.75
 MU=1.87
 T_OCEAN_INIT=289.28
 NYR_NON_CO2 = 21
 DIR_PATT = ''
 DIR_CLIM = ''
 DIR_ANOM = ''
 FILE_NON_CO2=.FALSE.
&END

&RUN
 CO2_INIT_PPMV=286.03
 FILE_SCEN_EMITS = 'emits_HADCM3.dat'
 FILE_NON_CO2_VALS=''
 FILE_SCEN_CO2_PPMV = 'co2_vals.dat'
 ANLG = .TRUE.
 ANOM = .TRUE.
 C_EMISSIONS = .TRUE.
 INCLUDE_CO2 = .TRUE.
 INCLUDE_NON_CO2 = .TRUE.
```

```
 LAND_FEED = .FALSE.
 OCEAN_FEED = .FALSE.
 NYR_EMISS = 241
&END
```

Table 34 Description of variables in the IMOGEN namelists

Variable name	Type and permitted values	Notes
Q2CO2	real	Radiative forcing due to doubling CO2 (W/m2)
F_OCEAN	real	Fractional coverage of the ocean
KAPPA_O	real	Ocean eddy diffusivity (W/m/K)
LAMBDA_L	real	Inverse of climate sensitivity over land (W/m2/K)
LAMBDA_O	real	Inverse of climate sensitivity over ocean (W/m2/K)
MU	real	Ratio of land to ocean temperature anomalies
T_OCEAN_INIT	real	Initial ocean temperature (K)
NYR_NON_CO2	integer	Number of years for which NON_CO2 forcing is prescribed
DIR_PATT	character	Directory containing the patterns. Files in this directory are expected to be in a specific format – see the IMOGEN example
DIR_CLIM	character	Directory containing initialising climatology. Files in this directory are expected to be in a specific format – see the IMOGEN example
DIR_ANOM	character	Directory containing prescribed anomalies. Files in this directory are expected to be in a specific format – see the IMOGEN example
FILE_NON_CO2	logical	If true, then non-CO2 radiative forcings are contained within a file
CO2_INIT_PPMV	real	Initial CO2 concentration (ppmv)
FILE_SCEN_EMITS	character	If used, file containing CO2 emissions. This file is expected to be in a specific format – see the IMOGEN example
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
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
ANLG	logical	If true, then use the GCM analogue model
ANOM	logical	If true, then incorporate anomalies
C_EMISSIONS	logical	If true, means CO2 concentration is calculated
INCLUDE_CO2	logical	If true, include adjustments to CO2 values
INCLUDE_NON_CO2	logical	If true, include adjustments to non-CO2 values
LAND_FEED	logical	If true, include land feedbacks on atmospheric CO2
OCEAN_FEED	logical	If true, include ocean feedbacks on atmospheric CO2
NYR_EMISS	integer	Number of years of emission data in file

6.18.1. Running IMOGEN and restrictions

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 spinup for the full transient run, and as such runs with IMOGEN must have no spinup (i.e. nspin=0 in INIT_OPTS). Since IMOGEN calculates the forcing for an entire year at once, an IMOGEN run must have a start time of 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 96x56 grid cells where each cell has size 3.75 degrees longitude by 2.5 degrees latitude with no Antartica. This means that:

- nxIn = 96 and nyIn = 56 in INIT_OPTS
- regLatLon = T, regLat1 = -55.0, regLon1 = -180.0, regDlat = 2.5 and regDlon = 3.75 in INIT_LATLON

The IMOGEN_EXAMPLE directory is provided as an example of setting up IMOGEN.

6.19. `INIT_DRIVE`: Meteorological driving data

```
>INIT_DRIVE

driveDataPer
ndriveFileTime, driveFilePer
readList
fileName
driveFileDate(1),driveFileTime(1)
driveFormat

ioPrecipType,l_point_data
tForSnow
tForConv,conFrac
io_rad_type,ioWindSpeed
useDiffRad,diffFracConst
z1_uv, z1_tq

ndriveExtra

>ASCBIN
byteSwapDrive
nfieldDriveFile
ndriveHeaderFile,ndriveHeaderTime,ndriveHeaderField
noNewLineDrive
>VARS
name(1)   fieldNumber(1)   interp(1)  nameFile(1)
name(2)   fieldNumber(2)   interp(2)  nameFile(2)
--- Repeat for each variable. ---
>ENDVARS

>NC
ncTypeDrive
>VARS
name(1)   SDFname(1)   nameFile(1)   interp(1)
name(2)   SDFname(2)   nameFile(2)   interp(2)
--- Repeat for each variable. ---
>ENDVARS
```

Table 35 Description of variables in the INIT_DRIVE section

Variable name	Type and permitted values	Notes
`driveDataPer`	integer 1 – 86400 (see notes)	The time step (seconds) of the driving data. This must be a multiple of the model timestep and must be at most 86400s (one day). 86400 must be a multiple of `driveDataPer`, so that data are read at the same times each day.
`ndriveFileTime`	integer >=1	The number of data files available for each variable, each file holding data for different times. If all variables

		are held together, this is the number of data files. If variables are held in separate files, this is the number of files for any one variable. If time templating is used (see Section 6.22), `ndriveFileTime` should be 1.
`driveFilePer`	integer	The period (seconds) of the files containing the driving data. This is only used if time templating is used (see Section 6.22). This must be at least as large as the period of the data (`driveDataPer`), and must be a multiple of the model timestep. Special cases: -1: monthly files -2: annual files
`readList`	logical	Switch controlling how the names of the files containing the driving data, and the times covered by each, are read. TRUE: names are read from another file FALSE: names are read from the run control file. This option is only allowed if `ndriveFileTime=1`.

`filename(1)`	character	If `ndriveFileTime=1` this is the name of the single data file (or the template name).
		If `ndriveFileTime>1`, this is the name of a file that lists the names and times of the data files. The first line of this file will be skipped (and so can be used for comments). All other lines are to be of the form:
		`filename, startDate,"startTime"`
		where
		`fileName` may contain variable-name-templating (see Section 6.22)
		`startDate` is in format `yyyymmdd`
		`time` is in format `hh:mm:ss`.

Starting time and date for first driving data file. Only used if `readList=FALSE` (otherwise these values are read from an external file).		
`driveFileDate`	integer	Date of first data in the driving data file, in format `yyyymmdd`.
`driveFileTime`	character	Time of day of first data in the driving data file in format `hh:mm:ss`. **It is recommended that all times entered in JULES use Greenwich Mean Time (GMT or UTC), not local time**. The time zone used here must match that under INIT_TIME (see Section 6.3).
`driveFormat`	character See Section 5.2.	Format of data files.
`ioPrecipType`	integer 1 to 4.	Flag indicating which precipitation variables are input, and how they are treated. (Note that all precipitation in JULES is considered to be either rainfall or snowfall.) 1: A single precipitation field is input. This represents the total precipitation (rainfall and snowfall). The total is partitioned between snowfall and rainfall using

		tForSnow (see below), and rainfall is then further partitioned into large-scale and convective components using tForSnow. Convective snowfall is assumed to be zero. 2: Two precipitation fields are input, namely rainfall and snowfall. The rainfall is partitioned between large-scale and convective, using tForConv (see below). Convective snowfall is assumed to be zero. 3: Three precipitation fields are input, namely large-scale rainfall, large-scale snowfall and convective rainfall. This cannot be used with l_point_data=TRUE. Convective snowfall is assumed to be zero, and tForSnow and tForConv are not used. 4: Four precipitation fields are input, namely large-scale rainfall, large-scale snowfall, convective rainfall and convective snowfall. This cannot be used with l_point_data=TRUE. tForSnow and tForConv are not used. Note that this is the only option that considers convective snowfall. 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..
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.
tForSnow	real >0	If ioPrecipType is 1 or 2, tForSnow 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.
tForConv	real >0	If ioPrecipType is 1 or 2, tForConv is the near-surface air temperature (K) at or above which the precipitation is assumed to be convective in origin. At lower temperatures, all the precipitation is assumed to be “large-scale” in origin. Also see conFrac.

		tForConv is not used if l_point_data is TRUE, since then there is no convective precipitation. tForSnow must be less than tForConv, implying that all solid precipitation is large-scale in origin (unless ioPrecipType=4, in which case tForSnow and tForConv are irrelevant).
conFrac	real >0	Convective precipitation covers the fraction conFrac of the gridbox.
io_rad_type	integer 1, 2 or 3	Flag indicating what radiation fluxes are input. 1: Downward fluxes of short- and longwave radiation are input. Normally this is the preferred option. 2: Downward shortwave and net (all wavelengths) downward radiation are input. The modelled albedo and surface temperature are used to calculate the downward longwave flux. 3: 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.
iowindSpeed	logical	Switch indicating how wind data are input. TRUE: the wind speed is input FALSE: the two components of the horizontal wind (e.g. the southerly and westerly components) are input.
useDiffRad	logical	Switch for input of diffuse radiation. TRUE: diffuse radiation is a time-varying input. Only allowed if io_rad_type=1 or 2. FALSE: diffuse radiation is set to a constant fraction (diffFracConst) of the total downward shortwave radiation.
diffFracConst	real	A constant value used to calculate diffuse radiation from the total downward shortwave radiation. Only used if useDiffRad=FALSE.
z1_uv	real >0.0	The height (m) at which the wind data are valid. This height is relative to the zero-plane *not* the ground.
z1_tq	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.
ndriveExtra	integer 0<ndriveExtra<= ndriveExtraMax	The number of "extra" (additional) driving variables that are to be input. These are additional to the variables that must be input. This facility has been added to provide the user with a relatively easy way to ingest new variables (that might be needed for a new development) with the minimal amount of coding. The maximum possible number of additional variables is determined by the parameter ndriveExtraMax, which is currently set to 10. Further details of each "extra" variable are provided below. Set to zero to turn off this facility (i.e. to provide no extra variables).

		See notes in Section 6.19.1.
>ASCBIN: If driveFormat='asc', 'bin' or 'pp':		
byteSwapDrive	logical	Switch controlling byteswapping of binary data. Only used if driveFormat='bin'. TRUE: the order of the bytes will be reversed after reading. This option allows data files written on a "little-endian" machine to be used on a "big-endian" machine, or vice versa. Some compilers have options that allow this behaviour. FALSE: no change will be made
nfieldFile	integer	Number of fields in each file.
nHeaderFile	integer >=0	The number of headers at the start of each file - see Section 5.2.
nHeaderTime	integer >=0	The number of headers at the start of each time - see Section 5.2.
nHeaderField	integer >=0	The number of headers at the start of field - see Section 5.2.
noNewLineDrive	logical	Switch describing format of an ASCII data file. TRUE: variables are arranged across one or more lines, and each variable does not necessarily start a new line. This option should be used if all the driving data for each time are one line of the input file (although it can also be used if the data are continued onto subsequent lines). FALSE: each variable starts on a new line. Only used if there is only one point in the input grid (and hence only one point in the model grid) and driving data are in ASCII files.
name	character	The name of the variable. This is used to identify the variable in the code, and is set in the code. Acceptable values are shown in Table 36. These must be entered exactly as listed in the table, and are case-sensitive.
fieldNumber	integer >=1	The field number in the file that holds data for this variable. See discussion of fields in Section 5.
interpFlag	character See Table 45.	Flag indicating how variable is to be interpolated in time
varNameFile	character	The substitution string used in the names of files that contain this variable. Only used if variable name templating is used in file names.
>NC: If driveFormat='nc':		
ncTypeDrive	character	Flag indicating the format (dimension names) of netCDF files. See Section 5.2.2.
name	character	See above under >ASCBIN.
SDFName	character	The name of the variable as used in a SDF. See discussion of SDF in Section 5.2.2.
varNameFile	character	See above under >ASCBIN.
interpFlag	character See Table 45.	See above under >ASCBIN.

The meteorological variables required by a run of JULES are determined by the choice of flags such as `ioPrecipType`. The variables that are listed must then match this expectation.

Table 36 Names of meteorological driving variables.

Name	Description	Comments
`diff_rad`	Diffuse radiation (W m^{-2})	Used if `useDiffRad`=TRUE.
`lw_down`	Downward longwave radiation (W m^{-2}).	Used with `rad_type`=1.
`lw_net`	Net downward longwave radiation (W m^{-2}).	Used with `rad_type`=3.
`sw_down`	Downward shortwave radiation (W m^{-2}).	Used with `rad_type`=1 or 2.
`sw_net`	Net downward shortwave radiation (W m^{-2}).	Used with `rad_type`=3.
`rad_net`	Net (all wavelength) downward radiation (W m^{-2}) .	Used with `rad_type`=2.
`precip`	Precipitation rate (kg m^{-2} s^{-1}).	Used with `ioPrecipType`=1.
`precipCR`	Convective rainfall rate (kg m^{-2} s^{-1}).	Used with `ioPrecipType`=3 and 4.
`precipCS`	Convective snowfall rate (kg m^{-2} s^{-1}).	Used with `ioPrecipType=4`.
`precipLR`	Large-scale rainfall rate (kg m^{-2} s^{-1}).	Used with `ioPrecipType`=3 and 4.
`precipLS`	Large-scale snowfall rate (kg m^{-2} s^{-1}).	Used with `ioPrecipType`=3 and 4.
`precipTR`	Rainfall rate (kg m^{-2} s^{-1})	Used with `ioPrecipType`=2.
`precipTS`	Snowfall rate (kg m^{-2} s-1).	Used with `ioPrecipType`=2 and 3.
`pstar`	Air pressure (Pa)	
`q`	Specific humidity (kg kg^{-1})	
`t`	Air temperature (K)	
`u`	Zonal component of the wind (m s^{-1}).	Used with `ioWindSpeed`=FALSE.
`v`	Meridional component of the wind (m s^{-1}).	Used with `ioWindSpeed`=FALSE.
`wind`	(Total) wind speed (m s^{-1}).	Used with `ioWindSpeed`=TRUE.
`ozone`	Surface ozone concentration (ppb)	Used with `l_o3_damage=TRUE`
`extraXX`	Additional driving variable (see ndriveExtra). XX should be replaced by 01, 02,…,min[`ndriveExtra`, `ndriveExtraMax`].	Used if `ndriveExtra` > 0.

6.19.1. Inputting extra driving variables

The facility to read in additional driving variables by setting `ndriveExtra`>0 is intended as a simple mechanism to allow access to additional data, particularly during model development. For example, a time-varying field of ozone concentration could be input after just a few lines of editing of the code. The additional variables must have the same frequency as the other variables and will be interpolated following the `interp` flags specified. The data can then be loaded into a new FORTRAN variable that the user has to create – this is best done in subroutine drive_update (look for comments containing "`iposExtra`"). The new variable itself could be provided via a module (e.g. `module forcing`).

6.19.2. Examples of specifying driving data

Example 1: single point driving data

In this example, we consider a case with one point in the input file, and all driving data for each time held on a single line of an ASCII input file. The input file is illustrated in Figure 4. The relevant entries in the run control file are shown below. Only the lines in **bold** are relevant, and irrelevant sections have been omitted.

```
>INIT_DRIVE
3600              !  driveDataPer
1,-9                  !  ndriveFileTime, driveFilePer
F                     !  readList
'data1.dat'           !  fileName
19970101,'00:00:00'   !  driveFileDate(1),driveFileTime(1)
'asc'                 !  driveFormat

2,T                   !  ioPrecipType,l_point_data
275.0                 !  tForSnow
375.0,0.2             !  tForConv,conFrac
1,T                   !  io_rad_type,ioWindSpeed
F,0.1                 !  useDiffRad,diffFracConst

10.0,10.0             !  z1_uv, z1_tq
0                     !  ndriveExtra

>ASCBIN
F          !  byteSwapDrive
10         !  nfieldDriveFile
1,0,0      !  ndriveHeaderFile,ndriveHeaderTime,ndriveHeaderField
T          !  noNewLineDrive
>VARS
pstar       9  nf  psfc    !  name,field,flag,name
t           6  nf  t
q          10  nf  q
wind        8  nf  u
lw_down     3  nf  lw
sw_down     2  nf  sw
precipTR    4  nf  liqp
precipTS    5  nf  solp
>ENDVARS
```

`ndriveFileTime`=1 indicates that all data are in one file.
`readList=FALSE` indicates that the name of the file is read from the run control file (not from a separate file).
`useDiffRad`=FALSE indicates that diffuse radiation is not input, rather it is calculated as 0.1 (the value of `diffFracConst`) of the total shortwave radiation.
`ndriveHeaderFile`=1 indicates that there is a single header line at the top of the file.
`noNewLineDrive=TRUE` shows that each variable is not on a new line (in fact all variables are on one line).
The entries following `>VAR` indicate where each variable lies in the input file. Note that we can skip the unrequired 'time' and 'obs1' fields in Figure 4.

```
Time   solar  long   rain   snow   temp           obs1   wind   press          humid
 1     3.3    187.8  0.0    0.0    259.10         83.0   3.610  102400.5       1.351E-03
 2     89.5   185.8  0.0    0.0    259.45         24.1   3.140  102401.9       1.357E-03
 3     142.3  186.4  0.0    0.0    259.85         56.9   2.890  102401.0       1.369E-03
----- data for later times ----
```

Figure 4. Lines of an example file of meteorological driving data in ASCII format.

Example 2: Driving data from binary files, one variable per file.
The relevant entries in the run control file are shown below. Only the lines in **bold** are relevant and irrelevant sections have been omitted.

```
>INIT_DRIVE

3600              !  driveDataPer
162,-9                !  ndriveFileTime, driveFilePer
T                     !  readList
'file_list.txt'       !  fileName
19820701,'03:00:00'   !  driveFileDate(1),driveFileTime(1)
'bin'                 !  driveFormat

2,F                   !  ioPrecipType,l_point_data
275.0                 !  tForSnow
298.2,0.3             !  tForConv,conFrac
1,F                   !  io_rad_type,ioWindSpeed
T,0.1                 !  useDiffRad,diffFracConst
10.0,10.0             !  z1_uv, z1_tq
2                     !  ndriveExtra

>ASCBIN
F          !  byteSwapDrive
1          !  nfieldDriveFile
0,0,0      !  ndriveHeaderFile,ndriveHeaderTime,ndriveHeaderField
T          !  noNewLineDrive
>VARS
pstar       1  nf  psfc   !  name,field,flag,name
t           1  nf  temp
q           1  nf  humid
```

```
u          1  nf  uwind
v          1  nf  vwind
lw_down    1  nf  long
sw_down    1  nf  solar
precipTR   1  nf  liqp
precipTS   1  nf  solp
diff_rad   1  nf  diffRad
extra01    1  nf  ozone
extra02    1  nf  co2
>ENDVARS
```

`ndriveFileTime`=162 indicates the number of files (for each variable).
`readList`=TRUE indicates that the names and times of each file are read from the file '`file_list.txt`'. The first few lines of this file are shown in Figure 5.

```
# List of meteorological data files. Columns are:
# file name, start date (yyyymmdd), start time (hh:mm:ss).
'met_data/%vv_data/%vv198207.dat', 19820701, '03:00:00'
'met_data/%vv_data/%vv198208.dat', 19820801, '03:00:00'
'met_data/%vv_data/%vv198209.dat', 19820901, '03:00:00'
------ rest of file not shown -----
```

Figure 5. Example list of driving data files using file name templating.

The presence of '`%vv`' in each file name shows that we are using variable name templating (see Section 6.22). The dates show that we in fact have monthly files (but note that we cannot use time templating for these files because the start time of 03H does not conform to the requirements described in Table 43). Furthermore, files for each variable are stored in separate directories. For example, skipping ahead to after `>VARS`, we see that the humidity variable is held in files such as '`met_data/humid_data/humid198207.dat`', while the surface pressure is held in the likes of '`met_data/psfc_data/psfc198207.dat`'.

The `ioPrecipType` value of 2 shows that we read in two components of precipitation: total solid and total liquid. The liquid is considered to be convective precipitation when the temperature is above `tForConv`, which here has a value of 298.2 K.

`useDiffRad`=TRUE indicates that diffuse radiation will be provided.
`byteSwapDrive`=FALSE indicates that the data will not be byteswaped after input.
`nfieldDriveFile`=1 shows that each data file contains a single field, which is consistent with the `field` number shown for each variable (all 1).
`ndriveExtra`=2 indciates that two additional, non-standard variables will be read in. These are listed as extra01 and extra02 in the list of variables. The filenames shown suggest that they are for ozone and CO_2, but they could represent any quantity that the user wants to input.

6.20. `INIT_IC`: Specification of the initial state

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 any other file. Another option is to prescribe a simple or idealised initial state, and this may be done via the run control file. It is also possible to set some fields using values from a file (e.g. a dump) but to set others using idealised values from the run control file (that is, effectively to override the values in the external file).

```
>INIT_IC

readFile
fileFormat (quoted)
dumpFile,allDump
fileName (quoted)
zrevSoil,zrevSnow
totalWetness
totalSnow

>ASCBIN
nheaderFile, nheaderField
>VARS
varName(1)   varFlag(1)   constVal(1)
varName(2)   varFlag(2)   constVal(2)
--- Repeat for each variable. ---
>ENDVARS

>NC
>VARS
varName(1)   varFlag(1)   constVal(1)   SDFname(1)
varName(2)   varFlag(2)   constVal(2)   SDFname(2)
--- Repeat for each variable. ---
>ENDVARS

# Data fields to be read from this file should appear below here.
>DATA
```

Table 37 Description of variables for `INIT_IC` section.

Variable name	Type and permitted values	Notes
`readFile`	logical	Switch controlling location of initial state data. TRUE: read from an external file (including a model dump) FALSE: read from the run control file.
`fileFormat`	character See Section 5.2.	Format of data. Only used if `readFile=TRUE`. Note that any dump file that is to be read (see `dumpFile`) can only be of type '`asc`' or '`nc`'.

dumpFile	logical	Switch indicating if file to be read is a model dump. Only used if readFile=TRUE. TRUE: the file is a model dump (restart) file that was written by this version of JULES. A dump file has known structure that can be navigated by JULES using header information. FALSE: the file is not a dump file
allDump	logical	Switch to allow easy use of all data in a dump file. Only used if dumpFile=TRUE, that is, if the file to be read is a model dump. TRUE: all variables required to initialise the run will be read from the given dump file. If a required field is not in the dump (e.g. if the dynamic vegetation model was not active in the earlier run but is now required), initialisation will fail and the run will stop. This option ignores all later input in the >ASCBIN and >NC sections. This is the easiest way to start from a dump file, as the user does not need to say what variables are to be found where – the model will look for all data in the dump file. FALSE: the information in the >ASCBIN and >NC sections is used to identify whether a field is to be read or set to a constant value, as usual.
filename	character	Name of file containing data. Only used if readFile=TRUE.
zrevSoil	logical	Switch indicating if soil data are stored in reverse order of levels. Not used if data are to be read from a dump file. TRUE: vertical order is reversed, with data stored in "bottom to top" order (i.e. bottom layer first) FALSE: standard vertical order, with data stored in "top to bottom" order (i.e. uppermost layer first)
zrevSnow	logical	Switch indicating if snow data are stored in reverse order of levels. Only used if nsmax>0. Not used if data are to be read from a dump file. TRUE: vertical order is reversed, with data stored in "bottom to top" order (i.e. bottom layer first) FALSE: standard vertical order, with data stored in "top to bottom" order (i.e. uppermost layer first)
totalWetness	logical	Switch controlling type of soil moisture data. Not used if soil wetness is to be read from a dump file. TRUE: soil wetness is prescribed as the total wetness (the sum of frozen and liquid components). FALSE: soil wetness is prescribed using two components (the frozen and liquid fractions separately).
totalSnow	logical	Switch controlling simplified initialisation of snow variables. Not used if snow data are to be read from a dump file. TRUE: only the total mass of snow on each tile (see

		snow_tile in Table 38) 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). This option can be used regardless of the value of `nsmax`. **If `nsmax`>0, this option is recommended as it means the user can avoid the complications of setting several snow variables in a consistent manner**. FALSE: all snow variables required for the current configuration must be input separately. The variables are listed in Table 38.
`>ASCBIN: The following are used if fileFormat='asc', 'bin', 'dump' or 'pp', or if readFile=FALSE.`		
`nheaderFile`	integer	The number of headers at the start of the file. See Section 5.2
`nheaderField`	integer	The number of headers at the start of a field. See Section 5.2
`varName`	character	The name of the variable. See Table 38.
`varFlag`	integer $\geq$-1	Flag indicating how the variable is initialised. Acceptable values: >0: The field number in the file that holds data for this variable. See discussion of fields in Section 5.1. If a dump file is being read, any integer $\geq$0 is accepted and then effectively ignored – this indicates that the field is to be taken from the dump and the exact field number is not required. -1: The field will be set to the value `constVal` (see below) at all points. This option can be used to specify an idealised initial condition.
`constVal`	real	The value to be used at all points. Only used if `flag=-1`.
>NC: The following are used if `fileFormat='nc'`.		
`varName`	character See Section 5.2.2	The name of the variable.
`varFlag`	integer $\geq$-1	Flag indicating how the variable is initialised. Acceptable values: >0: Default (effectively is ignored). -1: The field will be set to the value `constVal` (see below) at all points. This option can be used to specify an idealised initial condition.
`constVal`	real	See under >ASCBIN above.
`SDFvarName`	character	The name of the netCDF variable that is to be used.
`>DATA` If further initial data are to be read from the run control file (`readFile=FALSE`), these should now appear in the file, in the order indicated by the value of flag for each variable (see above). For example, if `tstar` is given a value of `flag=1`, and `cs` has `flag=2`, data for `tstar` and `cs` should then be listed, with each variable starting on a separate line.		

Some of these variables may not be required for a particular run, depending on the model configuration. The size of each variable is defined in terms of the following variables:

- land_pts - the number of gridboxes that contain any land.
- sm_levels - the number of soil layers.
- ntiles - the number of tiles at each gridbox.
- ntype - the number of surface types.
- npft – the number of plant functional types.
- nsmax – the maximum possible number of snow layers.
- n_olevs – the number of levels in IMOGEN ocean calculation
- nfarray – the size of IMOGEN array fa_ocean
- seed – the size of seed_rain in IMOGEN
- scalars

See Section 6.2 for further information about some of these variables.

Table 38 JULES variables that require to be specified to define the initial model state.

Note that this is a list of variables that have to be specifically listed in the input section. If all variables are to come from a model dump (allDump=TRUE), none of these variables needs to be listed. All variables names should be entered exactly as shown, including case.			
Name	Shape	Description.	Notes
General variables			
canopy	(land_pts, ntiles)	Amount of intercepted water that is held on each tile (kg m^{-2}).	Always required.
tstar_tile	(land_pts, ntiles)	Temperature of each tile (K). This is the surface or skin temperature.	Always required.
gs	(land_pts)	Stomatal conductance for water vapour (m s^{-1}).	Always required. This is used to start the iterative calculation of gs for the first timestep only.
Soil layer variables			
t_soil	(land_pts, sm_levels)	Temperature of each soil layer (K).	Always required.
sthuf	(land_pts, sm_levels)	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.	Only required if totalWetness =TRUE. Either sthuf or its components sthu and sthf are always required.
sthf	(land_pts, sm_levels)	Frozen soil wetness for each soil layer. This is the mass of	Only required if totalWetness =FALSE. Either sthuf or its components

		frozen water, expressed as a fraction of the water content at saturation. Note that the partitioning of water between liquid and solid fractions may be altered during initialisation. The procedure conserves the total water content, and uses the soil temperature (`t_soil`) to partition between the phases.	`sthu` and `sthf` are always required.
`sthu`	`(land_pts, sm_levels)`	Unfrozen soil wetness for each soil layer. This is the mass of unfrozen water, expressed as a fraction of the water content at saturation. See notes for `sthf` above.	Only required if `totalWetness` =FALSE. Either `sthuf` or its components `sthu` and `sthf` are always required.
Snow variables			
`snow_tile`	`(land_pts, ntiles)`	Amount of snow on each tile (kg m^{-2}).	Always required. . If `totalSnow` = TRUE, `snow_tile` holds the total snow mass on each tile. If `can_model`=4, this will be used to set the snow on the ground under the canopy. See Table 39 for further discussion.
`snow_grnd`	`(land_pts, ntiles)`	Amount of snow on the ground, beneath the canopy (kg m^{-2}), on each tile.	Only required if `can_model=4`. Not required if `totalSnow`=TRUE. A value should be given for all tiles, but it is only updated for tiles that refer to PFTs that have `snowCanPFT=1` (see Section 6.10).
`rho_snow`	`(land_pts, ntiles)`	Bulk density of lying snow (kg m^{-3}).	Only required if `totalSnow`=FALSE.
`rgrain`	`(land_pts, ntiles)`	Snow surface grain size (µm) on each tile.	Only required if `l_spec_albedo = TRUE`.

nsnow	(land_pts, ntiles)	The number of snow layers on each tile.	Only required if nsmax>0 and totalSnow=FALSE. Although this is an integer quantity, it is treated as a real number for convenience during input and output.
snowDepth	(land_pts, ntiles)	Depth of snow (kg m).	Only required if nsmax>0 and totalSnow=FALSE.
snowDs	(land_pts, ntiles,nsmax)	Depth of snow in each layer (kg m).	Only required if nsmax>0 and totalSnow=FALSE.
snowIce	(land_pts, ntiles,nsmax)	Mass of frozen water in each snow layer (kg m^{-2}).	Only required if nsmax>0 and totalSnow=FALSE.
snowLiq	(land_pts, ntiles,nsmax)	Mass of liquid water in each snow layer (kg m^{-2}).	Only required if nsmax>0 and totalSnow=FALSE.
tSnow	(land_pts, ntiles,nsmax)	Temperature of each snow layer (K).	Only required if nsmax>0 and totalSnow=FALSE.
rgrainL	(land_pts, ntiles,nsmax)	Snow grain size (µm) on each tile in each snow layer.	Only required if l_spec_albedo = TRUE and nsmax>0 and totalSnow=FALSE.
TOPMODEL variables			
zw	(land_pts)	Depth from the surface to the water table (m).	Only required if l_top=TRUE.
sthZw	(land_pts)	Soil wetness in the deep (“water table”) layer beneath the standard soil column This is the mass of soil water (liquid and frozen), expressed as a fraction of the water content at saturation.	Only required if l_top=TRUE.
Soil and vegetation carbon variables			
cs	(land_pts,dim2) See notes for dim2.	Soil carbon (kg m^{-2})	Always required. dim2=1 if TRIFFID is not being used (l_triffid=FALSE), in which case the total soil carbon is input. dim2=4 if TRIFFID is being used, to hold the 4 pools of the RothC model

			(biomass,humus,decomposable plant material and resistant pant material). Note that `cs` is a prognostic (time-evolving) variable only if TRIFFID is selected.
`frac`	`(land_pts, ntype)`	The fraction of land area of each gridbox that is covered by each surface type.	Always required, but can be read at `INIT_FRAC`. This variable has to be set either in this section of the run control file, or in the section tagged `INIT_FRAC` (see Section 6.5). If `l_veg_compete=TRUE` (see Section 6.2), `frac` must be set here, as part of the initial condition (e.g. from a model dump). If `l_veg_compete=FALSE` (i.e. the fraction of each type is static), the fraction may be set here, as part of the initial condition, or in `INIT_FRAC`. The switch `readFracIC`, described in that section, is important in this case.
`lai`	`(land_pts, npft)`	Leaf area index of each PFT.	Only initialised here if phenology is switched on in `INIT_OPTS` (see Section 6.2). If phenology is off, LAI is not a prognostic variable and it is initialised in either `INIT_VEG_PFT` or `INIT_VEG_VARY`.
`canht`	`(land_pts, npft)`	Height (m) of each PFT.	Only initialised here if TRIFFID is switched on in `INIT_OPTS` (see Section 6.2). If TRIFFID is off, canht is not a prognostic variable and it is initialised in either `INIT_VEG_PFT` or `INIT_VEG_VARY`.
IMOGEN variables			
`co2_ppmv`	`scalar`	Atmospheric CO2 concentration (ppmv)	Required only if IMOGEN is enabled
`co2_change_ppmv`	`scalar`	Gradient of atmospheric CO2 concentration (ppmv/yr)	Required only if IMOGEN is enabled
`dtemp_o`	`(n_olevs)`	Ocean temperature anomalies (K)	Required only if IMOGEN is enabled
`fa_ocean`	`(nfarray)`	CO2 fluxes from the atmosphere to the ocean (ie positive upwards) (ppm/m2/yr)	Required only if IMOGEN is enabled

seed_rain	(seed)	Seeding number for subdaily rainfall.	Required only if IMOGEN is enabled
cv	(land_pts)	Gridbox mean vegetation carbon (kg C m^{-2})	Required only if IMOGEN is enabled

Note that it might appear that nsmax>0 requires an excessive number of variables, some of which are redundant. However, many of the details as to why all these variables must be input relate to subtleties and the needs of implementation in the Unified Model (weather forecast and climate model). It is true that the values of several of these variables must be consistent (e.g. snow depth and snow depths in layer), and totalSnow=TRUE is useful in allowing a simple initialisation.

Table 39 Further details of snow variables

Name	Description	Required if nsmax=0?	Required if totalSnow =TRUE?	Notes
snow_tile	Mass of snow	Y	Y	If can_model≠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 snowCanPFT=1), snow_tile is interpreted as the snow on the canopy, except as overridden by totalSnow=TRUE. If totalSnow=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).
snow_grnd	Mass of snow on ground under canopy	Y	N	Only required if can_model=4. If totalSnow=T this is set to snow_tile at tiles where can_model=4 is active, to zero at all other tiles.
rho_snow	Bulk density of lying snow	Y	N	If totalSnow=T, this is set to rho_snow_const.
rgrain	Surface grain size	Y	Y	Only required if l_spec_Albedo=TRUE.
nsnow	Number of layers	N	N	If totalSnow=T this is calculated from the snow depth.
snowdepth	Depth of snow	N	N	If totalSnow=T, this is calculated from mass and density of snow.
snowDs	Depth in each layer	N	N	If totalSnow=T this is calculated internally.
snowIce	Ice content in each layer	N	N	If totalSnow=T all snow is assumed to be ice.
snowLiq	Liquid content in each layer	N	N	If totalSnow=T this is set to zero.

tSnow	Temperature in each layer	N	N	If totalSnow=T this is set equal to the temperature of the top soil layer.
rgrainL	Grain size in each layer	N	N	Only required if l_spec_Albedo=TRUE. If totalSnow=T this is set to rgrain.

6.20.1. Examples of specification of initial state

Example 1: A single point, state from the run control file

In this example, we consider a run at a single point and read all data from the run control file. The relevant entries in the run control file are shown below. Only the lines in **bold** are relevant and irrelevant sections have been omitted. Assumptions include that nsmax=0, l_triffid=FALSE.

```
>INIT_IC

F                                ! readFile
'asc'                            ! fileFormat (quoted)
F,F                              ! dumpFile,allDump
'a0001_dump.19970105'            ! fileName (quoted)
F,F                              ! zrevSoil,zrevSnow
T                                ! totalWetness
T                                ! totalSnow

>ASCBIN
0,0                        ! nheaderFile,nheaderField
>VARS
sthuf        1     0.9   !  varName,varFlag, constVal
canopy       2     0.0
snow_tile    3     0.0   ! Note that none of these "constVal"
tstar_tile   4     0.0   ! values are used in this case (because
t_soil      5      0.0   ! varFlag≠-1). Instead, values
cs           6     0.0   ! are listed after >DATA.
gs           7     0.0
>ENDVARS

# Data fields to be read from this file should appear below here.
>DATA
 0.749, 0.743, 0.754, 0.759  !  sthuf
9*0.0                        !  canopy
9*0.0                        !  snow_tile
9*276.78                     !  tstar_tile
276.78,277.46,278.99,282.48  !  t_soil
12.100                       !  cs
0.0                          !  gs
```

readFile=FALSE indicates that all data will be read from the run control file; no other file is involved and several of the following lines are not used. In this case, we use the >ASCBIN section to describe the data.

The seven variables that are required to initialise this particular run are then listed. The second entry in each line gives the position in the input data for each field. Since all the data are to be read from the run control file, which is easily edited, it is easiest to list these variables in the order in which the data will be presented (i.e. field numbers should be 1, 2, 3,…). In this example, all the field numbers are >0, indicating that the data will be read from the >DATA section (and that the constVal entries will be ignored).
Note that data for soil variables are presented in the order "top to bottom", i.e. surface layer first.

Example 2: Initial state specified as a mixture of spatial fields and constant values
In this example, we consider a run at a single point and read all data from the run control file. The relevant entries in the run control file are shown below. Only the lines in **bold** are relevant and irrelevant sections have been omitted.

```
>INIT_IC

T                                 !    readFile
'bin'                             !    fileFormat (quoted)
F,F                               !    dumpFile,allDump
'a001_initial_state.gra'          !    fileName (quoted)
F,F                               !    zrevSoil,zrevSnow
T                                 !    totalWetness
T                                 !    totalSnow

>ASCBIN
0,0                         !  nheaderFile,nheaderField
>VARS
sthuf        7     0.9   !  varName,varFlag, constVal
canopy      -1     0.0
snow_tile   -1     0.0
tstar_tile  -1   275.0
t_soil      -1   278.0
cs          -1    10.0
gs          -1     0.0
>ENDVARS
```

readFile=TRUE indicates that the binary file "a001_initial_state.gra" will be used to set the initial state (for some variables).
The seven variables that are required to initialise this particular run are then listed. The second entry in each line gives the position in the input data for each field. For most variables, the value -1 indicates that the field is to be initialised as spatially constant using the value given under constVal. For example, the temperature in each soil layer (t_soil) will be set to 278K at all locations in the model grid. For soil wetness (sthuf), the field number is given as 7 – meaning that soil wetness will be set using the data starting at field 7 in the named input file. Since zrev=TRUE, these data are stored in the file in "non-standard" order (i.e. bottom to top), so that field 7 is the deepest layer (and, assuming 4 soil layers, field 10 will be used for the uppermost layer).

Example 3: Initial state specified 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 variables. Consider a run at a single point and read all data from the run control file. The relevant entries in the run control file are shown below. Only the lines in **bold** are relevant and irrelevant sections have been omitted.

```
>INIT_IC

T                               !    readFile
'nc'                            !    fileFormat (quoted)
T,T                             !    dumpFile,allDump
'a001_dump.nc'                  !    fileName (quoted)
```

`readFile`=TRUE indicates that the netCDF file "`a001_dump.nc`" will be used. dumpFile=T indicates that this is a dump file from an earlier run, and `allDump`=T indicates that all variables are to be set using values from the dump file and therefore all subsequent entries in the `INIT_IC` section of the run control file are ignored.

6.21. INIT_OUT: Specification of output from the model

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 from the entire model grid to one file, and high-frequency output from a subset of points to another file.
- 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.

This flexibility comes at the expense of having to set several values in the run control file. However, default values allow the user to select certain configurations relatively easily.

The first values in this section of the run control file concern general details of the output, such as the file format, that apply to all output profiles. This is followed by a separate section for each output profile, describing the variables, the grid and time sampling for that profile.

6.21.1. INIT_OUT: General values related to output

This section starts with the tag `>INIT_OUT`.

```
>INIT_OUT

run_id
outDir

dumpFreq
dumpFormat
dumpStatus

nout
outFormat
gradsNc
outStatus
yrevOut
zrevOutSoil,zrevOutSnow
numMonth
useTemplate
undefOut
zsmc,zst
outEndian
```

Table 40 Description of variables in the `INIT_OUT` section.

Variable name	Type and permitted values	Notes
`runID`	character*10	A name or identifier for the run. This is used to name output files and any model dumps.
`outDir`	character*150	The directory used for output files. This can be an absolute or relative path. Enter "." to write output to the directory from which JULES is run.
`dumpFreq`	`integer` 0 to 4	Flag indicating how often the model state is to be 'dumped' (written to a file). Acceptable values are: 0: no dumps are written 1: only the final state of the model (at the end of the integration) is dumped 2: dump initial and final model states 3: as 2 but also write a dump at the end of the spin-up phase 4: as 3 but also write a dump at the end of each calendar year. A model dump captures the state of the model at a given point in the integration. If a final dump is saved, the integration can later be extended by starting another run from this final dump. For long integrations, or large domains, it is recommended that dumps are saved for every year, so that in the event of any trouble such as a model crash, the integration can be completed without having to start again from the initial state. NB A run that is carried out in several steps, each starting from the model dump for the previous step, will generally not evolve identically to a single run that proceeds without the intermediate dumps. This is due, in part, to a loss of precision when the model state is written to the dump file.
`dumpFormat`	character 'asc' or 'nc'	Format for dump files. ASCII or netCDF.
`dumpStatus`	character 'new' or 'replace'	The file status used when writing a model dump. Acceptable values are: '`new`' – if a file with the same name already exists, the run will terminate. '`replace`' – if a file with the same name already exists, it will be overwritten.
`nOut`	integer	The number of output profiles. Each profile generates a separate stream of data, as explained above.
`outFormat`	character	The format for output files. Acceptable values are: '`asc`': ASCII files '`bin`': flat binary files '`nc`': netCDF files
`gradsNc`	logical	Switch controlling details of netCDF output files.

		Only used if `outFormat`='nc'. TRUE: netCDF output will be constructed so as to be readable by GrADS. In particular, snow layer variables will be split so that each tile is represented with a separate variable (otherwise there are too many dimensions for GrADS to cope with). FALSE: netCDF output might not be readable by GrADS (but in many cases is).
`outStatus`	character 'new' or 'replace'	The status used when opening files. This is the value given to the FORTRAN "status" argument of an OPEN statement [e.g. `open(1,status='new')`], or the equivalent for netCDF files. '`new`': file must not already exist. If the code tries to create a file with the same name as an existing file, the run will terminate. '`replace`': If the file exists, delete it and replace with a new version.
`yrevOut`	logical	TRUE: reverse the order of the rows in the output, so that these are written in "North to South" order. FALSE: use the default "South to North" order, with the southernmost row of data being the first in the file.
`zrevOutSoil`	logical	Switch indicating if soil layer data are to be output in reverse order of levels compared with JULES's default. TRUE: reverse the order of the vertical levels in the output, so that these are written in "bottom to top" order (i.e. bottom layer first). FALSE: use the default "top to bottom" order (i.e. top layer first).
`zrevOutSnow`	logical	Switch indicating if snow layer data are to be output in reverse order of levels compared with JULES's default. TRUE: reverse the order of the vertical levels in the output, so that these are written in "bottom to top" order (i.e. bottom layer, closest to soil, first). FALSE: use the default "top to bottom" order (i.e. top layer first).
`numMonth`	logical	Switch controlling the date format used in file names. TRUE: months are represented by the numbers 1 to 12. FALSE: months are represented by 3-character strings (jan, feb, mar,…)
`useTemplate`	logical	This relates to GrADS files (generated by `outFormat='bin'` or `'nc'`). Switch to activate the writing of template '`.ctl`' files. A template ctl file allows GrADS to access several data files via one ctl file. TRUE: all suitable ctl files will use the template

		option. FALSE: generate a separate ctl file for each data file. Note: A template ctl file will not be able to describe the data if there are any missing times at the start of a file – this is a limitation of the current JULES code, rather than GrADS. For example, if daily data are to be written to monthly files, with a template ctl, but the run starts midway through the month, JULES will only write output data for the latter part of the month. GrADS will look for data for all days in the month, but not be able to find them, so the user will not be able to plot the first month.
`undefOut`	real	The value written to output files to represent "missing" or "undefined" data.
`zsmc`	real	If a depth-averaged soil moisture diagnostic is requested, the average is calculated from the surface to this depth (m).
`zst`	real	If a depth-averaged soil temperature diagnostic is requested, the average is calculated from the surface to this depth (m).

`outEndian`	`character`
	`'little_endian'`
	or `'big_endian'`
		Only used for GrADS output files (`outFormat='bin'`), this describes the byte ordering of the computers on which JULES is run. It is only included in the 'options' line of GrADS ctl files, i.e., in metadata describing the file. It does NOT alter the byte order of the output.
		Acceptable values are:
		'`little_endian`' – for little endian computers (e.g. PCs)
		'`big_endian`' – for big endian computers (e.g. Suns)

6.21.2. `NEWPROF`: details of each output profile

This section starts with the tag `>NEWPROF`.

Each of the `nout` output profiles requires a section that describes that profile, such as the times when output is to be generated, which points are to be output, which variables are to be output, and more. The size of a regular latitude/longitude gridbox (input as `regDlat`, `regDlon` in control file – see Section 6.4.3) is also used as the size of a gridbox in the output.

```
>NEWPROF

outName
outPer,outFilePer
outSamPer
outDate(1),outTime(1)
```

```
outDate(2),outTime(2)

pointsFlag(1:2)
outAreaLL
outRangeX(1:2),outRangeY(1:2)
outCompress,outLLorder

readFile
fileName

pointsOut
mapOut(1:pointsOut,1)
mapOut(1:pointsOut,2)

>GRID
outGridNx,outGridNy

>VARS
flag name useName
--repeat for each output variable --
>ENDVARS
```

Table 41 Description of variables for each output profile.

Variable name	Type and permitted values	Notes
`outName`	character (len=10)	The name of this 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.
`outPer`[9]	integer	The period for output (seconds). This must be a multiple of the timestep length (except for the special cases <0 given below). It must not exceed 30 days (2592000 seconds), except for the special cases. Special cases: `0`: generate output every timestep. `-1`: monthly period `-2`: annual period (calendar years)
`outFilePer`[9]	integer	The period for output files (seconds), i.e. the time interval within which all output goes to the same file. This must not exceed 30 days (2592000 seconds), except for the special cases given below. The file period must be consistent with the output period (e.g. we can't have daily files for monthly output). Output may be generated for only part of a run (see `outDateStart` below), and `outFilePer` controls how the data

[9] Many variables that are input in terms of seconds (such as `outPer` and `outFilePer`) are converted within the code to a number of model timesteps.

		are stored during that part of the run when the output is "active". Special cases: `0`: output is every timestep, and a new file is created every timestep `-1`: monthly files (all output for a month goes to the same file) `-2`: annual files (calendar years) `-7`: all output goes to one file, but each cycle of spin up creates a separate file `-8`: all output goes to one file, but all output during spin up goes to a separate file `-9`: all output (for all times) from this profile goes to one file
`outSamPer`[9]	integer	The sampling period (seconds) for time-averages and accumulations. This must be a factor of the output period (`outPer`). Special case: 0 means sample every timestep. **The recommended setting is `outSamPer=0`.** However, 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. Remember that if fields are not sampled every timestep, the output averages will only be approximations.
`outDateStart`	integer	Date in format yyyymmdd. Output from this profile is first generated at the date and time indicated by `outDateStart` and `outTimeStart`. These must be within the "main run", except for the special cases noted below. Note that output is only generated at the end of a timestep, except for the special cases noted below. Special cases for `outDateStart`: 0: output all times through the run, including any spin-up[10] -1: output at all times after spin-up is complete -2: output only at the start of the first timestep of the run (used to output the initial state only). Note that, at present, the only time at which output can be generated at the *start* of a timestep is at the start of the run, when `outDateStart=-2` will output the initial state. Thus the only way in which the initial state can be output is to have an output profile with `outDateStart=-2`. All output at later times then has to be generated via another output profile. (This is a slight oversimplification – see footnote 10!) Note (a complication that you can ignore, but to really understand

[10] Under some circumstances, `outDateStart=0` will also output the initial state of the model. These circumstances are that the period of the output equals the timestep (i.e. information for every timestep) and that all output goes to a single file (`outFilePer=-9`). The timestamp information included with the output allows the user to determine whether this initial state has been output.

		your output you might need to follow this!): For time-averaged output, `outDateStart` and `outTimeStart` specify the first time at which data will be included in the accumulation that is used to calculate the average (call this time t1). If t1 happens to be a time when any earlier average would be complete (i.e. had the output been started earlier, an average would have been calculated at t1), the average cannot be calculated at this start time and a "missing value" is output. Example: Hourly averages starting at midnight 1st Jan 1996, and using a model timestep of 1800s (`outPer` = 3600, `outPutDateStart` = 19960101, `outTimeStart` = 00:00:00). At midnight 1st Jan 1996, this output stream is "activated". The code then realises that the average over the previous hour should be calculated immediately (because an hourly average is always calculated "on the hour"), but because sufficient times have not been accumulated, the first value of this average (representing the average over the hour ending at midnight) is set to "missing". The first "good" value will be the average ending 01H. On the other hand, instantaneous values *can* be output at 0H in this case because there is no need to accumulate any earlier values.
`outTimeStart`	character *8	Time of day (in format hh:mm:ss) at which output begins. Not used if `outDateStart` is one of the special cases.
`outDateEnd`	integer	Date on which output ends. Not used if `outDateStart` is one of the special cases.
`outTimeEnd`	character *8	Time of day at which output ends. Not used if `outDateStart` is one of the special cases.
`pointsFlag(1)`	integer 0, 1, 2	Flag indicating how the points to be output are selected. 0 = all points in the model grid will be output 1 = points in a rectangular subsection will be output. 2 = the points to be output will be listed individually
`pointsFlag(2)`	integer 0 to 5	Flag indicating how the locations in the output grid of output points will be calculated. 0: the output grid will be the model grid 1: the output grid will be the rectangular subsection specified via `pointsFlag(1)=1`. This option can only be used in conjunction with `pointsFlag(1)=1`. 2: the location of each output point will be listed individually. This option can only be used in conjunction with `pointsFlag(1)=2`. 3: the output grid will be the smallest rectangle that contains all the output points. This option requires that the model grid is rectilinear (or is a subset of such a grid). 4: the output grid will be the same as the input grid. 5: the output grid is a vector. This option can only be used in conjunction with `pointsFlag(1)=2`. In this case, the points to be output were specified by reading a list and they are simply written in the same order to an output vector. This option can be useful if a disparate set of points from an irregular grid has been selected for output, and saves having to specify a trivial mapping via `pointsFlag`(2)=2.

		Depending upon the shapes of the input and model grids, it may be possible to produce the same output grid via different combinations of the values of `pointsFlag`. Similarly, certain combinations will be less useful for particular grids.
`outAreaLL`	logical	Switch indicating how to interpret the coordinates `outRangeX` and `outRangeY`. Only used if `pointsFlag(1)=1`. TRUE: co-ordinates are longitude and latitude. FALSE: co-ordinates are x and y indices (column and row numbers).
`outRangeX(1:2)`	real array	x-coordinates of the sub-area to be output. Depending on `outAreaLatLon`, these are longitudes (in range -180 to 360º) or column numbers. Only used if `pointsFlag(1)`=1. Column numbers are those in the INPUT grid. If values are column numbers, the code uses the nearest integer to the input value.
`outRangeY(1:2)`	real array	As `outRangeX`, expect in latitudinal (y) direction.
`outCompress`	logical	Switch indicating if output data are to be "compressed" so that only model points are output. TRUE: Only output model points. Also output the mapping between the model points and the output grid (e.g. how to scatter the output points across a larger grid). The mapping is output in a form suitable for use with GrADS' pdef. FALSE: If the output grid is larger than the number of points to be output, the grid is filled with "missing data" or padding values. See Section 6.21.3 for further discussion of output compression. If the output grid is the same size as the number of points to be output (so no compression is possible), `outCompress=TRUE` may still cause output to differ in format from `outCompress=FALSE` (the points may be written in a different order), so `outCompress` should always be set to FALSE unless needed otherwise. Note that if `outCompress=TRUE`, then `yrevOut` is ignored for the profile (it becomes irrelevant).
`outLLorder`	logical	Switch indicating the coordinate system to be used to determine the locations of the output points in the output grid. Only used if `pointsFlag(2)`=1 or 3. TRUE: use the latitude and longitude of each point to determine its location in the output grid. FALSE: use the row and column number in the INPUT grid to determine where each point goes in output grid. This option is particularly useful if the input grid is rectilinear but is not regular in latitude and longitude (e.g. it could be a rotated grid).

		The output can then be placed on the same rectilinear grid.
`readFile`	logical	Switch controlling location of output mapping. Only used if `pointsFlag(1)=2` (i.e. a mapping is to be input). TRUE: read from an external file FALSE: read from the run control file.
`filename`	character	The name of the file that contains output mapping. Only used if `readFile=TRUE`.
`pointsOut`	integer 1 to size of grid	The number of points to be output. This is only used if `pointsFlag(1)=2`.
`mapOut(1:pointsOut,1)`	integer array	A list of the points that are to be output. The list gives the locations (point numbers) in the INPUT grid (which need not be the same as the model grid). Only used if `pointsFlag(1)=2`.
`mapOut(1:pointsOut,2`	integer array	A list giving the destination (location in output grid) for each output point. The list gives the point number in the output grid. Only used if `pointsFlag(2)=2`
`outGridNx`	integer	Number of columns in the output grid. This is the full, uncompressed output grid. If compression is applied, the actual output may be smaller, but can be scattered across a grid with this number of columns. Only used if `pointsFlag(2)=2`, in which case the user specifies all aspects of the output grid and mappings. Otherwise the size of the output grid is calculated by the model.
`outGridNy`	integer	As `outGridNx`, but number of rows.
`>VARS` `A list of variables to be output is provided between the tags >VARS and >ENDVARS.`		
`flag`	character*1 S, M or A	Flag indicating type of processing. Acceptable 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 `outPer`. For time-accumulation variables, `outPer` 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 set via `outSamPer`. NB A time-accumulation is initialised at the start of a run (actually at the start of each section of a run so that it is reinitialised after any spin up is completed – see Section 6.3.3) and thereafter accumulates until the end of the run

		(actually to the end of each section of a 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.
`name`	character	The name of an output variable (one word). This is the internal name as used in the model code. A list of available variables is provided in Section 9. This list was correct at the time of writing, but the most reliable way to determine exactly which variables are available for a particular version of JULES is to look at the variables listed in the subroutine `init_out_varlist`, and which can be echoed to screen at the start of a JULES run by setting `echo=TRUE` in `INIT_OPTS` (see Section 6.2). A variable may appear more than once in an output profile, as long as each time it appears with a different time flag – e.g. instantaneous and time-average values.
`useName`	character	The name to be used in the output (one word). This variable need not be specified. If `useName` is not provided, the code will substitute `name` instead. This facility allows the user to choose to call output variables by names other than those used in the code, for example to use names that are more memorable, or shorter names to avoid typing! Although the name should be a single word, characters such as underscore ("_") may be used.

6.21.3. Compression of the output grid

As noted above, `outCompress=TRUE` can be used to compress the output data so that any "missing" points are not written and file size is reduced. Although this facility was designed to work with the pdef option of GrADS, it might be useful with other packages too, with the proviso that the user may have to tell another package how to use the available information.

This facility will be described by considering an example in which we have global input data on a 1° grid, and JULES is run at land points only. We would like to visualise the output plotted on the full globe. The input grid is of size 360×180=64800 points, of which only about 25% are land points at which the model is run. If we set `outCompress=TRUE`, the output files will contain data only for the land points, and a mapping is defined so that the land points can be plotted in their correct positions on the Earth. This leads to considerable saving on disc space. The data in the output file is written as a vector (of ~15000 points in this case), in the order that they are held in the model grid. The mapping is written to a binary file that contains 3 fields on the full, expanded grid (360x180 points in this example, starting from the southwest corner, proceeding across each row, then onto next row – i.e. the default JULES order). The first field is integer, and gives the location in the output vector (of ~15000 points) that should be plotted at this location in the globe. If there are no data for a point (i.e. a sea point in this case), the missing data value is inserted. The second field is real, and is 1.0 at points with data, elsewhere 0.0. The third field is not used by JULES (it deals with wind rotation) and will consist of the missing data value.

GrADS' pdef option can be used to display just such a thinned grid, i.e. the "full" grid is populated with values from the "thinned" grid, with missing data values inserted at all other points. Note that `outCompress` as implemented in JULES is a subset of the full pdef available in GrADS, namely where pdef is used with a supplementary file, and each point in the "thinned" output grid maps onto a single point in the "full" grid – effectively there is no interpolation. Thus the latitudes and longitudes of the model gridpoints (specified in `INIT_GRID` above) must be consistent with those specified here for the "full" grid.

If a package other than GrADS is being used to display the thinned data, the user will have to either work out how to use the GrADS mapping between the vector and the full grid, or create new mapping data.

6.21.4. An example of output grids and mapping

This example uses the grids shown in Figure 6. The model grid has `nx`=5, `ny`=4 as shown, and is regular in latitude and longitude. For simplicity, we will assume that the input grid was identical to the model grid. The user wishes to output the 3 shaded points to an output grid with `nxOut`=2, `nyOut`=2, maintaining their relative positions (as given by latitude and longitude).

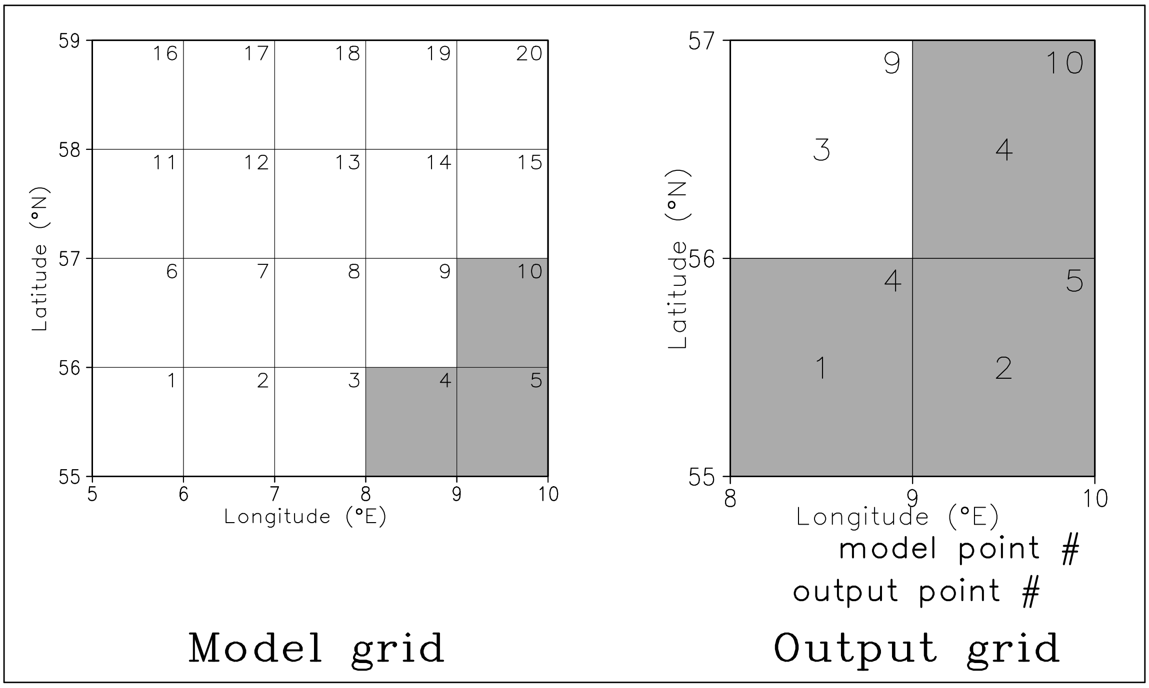

Figure 6. An example of the grids used in output mapping.

The easiest way to achieve this is to use the following lines in the run control file (irrelevant lines have been omitted):

```
2,3            !  pointsFlag(1:2)
F,T            !  outCompress,outLLorder
F              !  readFile

3              !  pointsOut
```

```
4,5,10        !  mapOut(1:pointsOut,1)
```

`pointsFlag(1)=1` means that the chosen points will be listed individually.
`pointsFlag(2)=3` means the output grid is to be the smallest rectangle that includes all output points.
`outCompress=F` means the output grid will be padded as necessary. In this case, it means that output point #3 in Figure 6 will be filled with the missing data value.
`outLLorder=T` means the location of each point in the output grid is calculated using the latitude and longitude of the point.
`pointsOut=3` indicates that 3 points are to be output.
`mapOut(1:pointsOut,1)` indicates that the points to be output are numbers 4, 5 and 10 in the input grid (which is identical to the model grid in this case).
The model uses the latitude and longitude of each point to establish that the chosen points should occupy locations 1, 2 and 4 in the output grid, and that location 3 should be filled with the missing data flag (`undefOut`).

The same effect could be achieved by using `pointsFlag(1)=2`, `pointsFlag(2)=2`, `mapOut(:,2)=1, 2, 4`, `outGridNxy=2, 2`, i.e. the user can completely specify the mapping and grid shape. Calculating `mapOut(:,2)` is trivial in this example, but would involve the user in more and unnecessary work if many more points were to be output.

6.21.5. Notes on output

1. A warning is raised if any output is not generated because the output interval is not completed. This can occur when a run starts or ends partway through an output period, or if a spin up cycle ends partway through an output period. For example, if monthly average diagnostics are requested, but the run ends on the 10th day of a month, the final monthly average is incomplete. In such cases, a value is still written to the output file, but the details of this value vary between cases. In short, a monthly or annual average is calculated if a "large fraction" of the month or year has been simulated, but averages over shorter periods are not calculated and a "missing data" value is output. For details, see the code.
2. GrADS output: A control file (`.ctl` file), that describes GrADS output, includes a specification of the number of times of output that are contained in the associated data files (the TDEF line). When a data file is first opened, a control file is written, with an estimate of the expected number of times that will be written. Sometimes this initial estimate will prove wrong (for example, if the model is spinning up the number of spin up cycles may not be known in advance), and the `.ctl` file is later rewritten when the data file is complete. Under most circumstances, this procedure is carried out without any problem. However, if the user opens the `.ctl` file in GrADS while the integration is still underway, it may not correctly specify the number of times. In that case, the `.ctl` file will be correct if reopened at a later time. However, if the user has moved the `.ctl` file while the integration is underway, it cannot be rewritten and a warning is raised if an attempt is made to rewrite it.
3. Driving data, such as meteorological or vegetation data, may not be correctly represented in output at the start of the first timestep of the run (i.e. time=0), depending upon the frequency of data and any temporal interpolation. The problem arises because the initial output is generated before the procedures that update the driving data are called. Under some circumstances, the driving data will already have been updated during the

initialisation, and so the output will be correct. In other cases, the initial output will have "nonsense" values such as zero for the driving data.

4. The code that generates output contains many options and has to deal with a variety of possibilities in terms of output frequency, run dates, spin up and the likes. Until the code has been thoroughly tested by the user community, early versions of JULES are quite likely to contain bugs, particularly in the output code. If a user finds an error with the output, the bug should be reported, but in the meanwhile JULES will hopefully run correctly if "simpler output" is requested. Two simplifying options, that may not always be practicable for the user, are to request snapshot diagnostics (rather than time averages; in cases of extreme difficulty these snapshots should be every timestep), and to send all output to a single file.

6.22. 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[11]. 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 42. These are 3-character strings, starting with "%". Note that any file name that contains "%" is assumed to use templating.

Table 42 Valid substitution strings for substitution templates.

Substitution string	Description
Time templating	
`%tc`	1-character representation of decade (Met Office files)
`%y4`	4-digit year
`%y2`	2-digit year
`%yc`	1-character representation of year (Met Office files)
`%m2`	2-digit month
`%m1`	1- or 2-digit month
`%mc`	3-character month abbreviation
`%mm`	1-character representation of month (Met Office files)
`%d2`	2-digit day of month
`%d1`	1- or 2-digit day of month
`%dm`	1-character representation of day of month (Met Office files)
`%h2`	2-digit hour of day
`%h1`	1- or 2-digit hour of day
`%hc`	1-character representation of hour of day (Met Office files)
`%n2`	2 digit minute (leading zero if needed)
Variable name templating	
`%vv`	A character variable

[11] JULES templating is similar to that used by GrADS, with a few important differences. JULES only allows a subset of the GrADS substitution strings (not including the %ch string used with chsub), but is more flexible in how it deals with time-templating.

6.22.1. Time templating

Information about the time of each file is contained in the file name. Valid substitution strings are listed in Table 42 and examples of the use of time templating are given in Table 44.

The substitution template must be compatible with the period (frequency) of the data files. If a substitution template includes a substitution string that refers to a period of a day or longer, each file must contain data for no more than one period. For example, if `%m2` appears in the template, each file must contain data from at most one calendar month. For periods less than one day (i.e. hours and minutes), data for more than one period can be held in the same file, but the file period must be a factor of one day[12].

The start time of each file must also follow (slightly complicated) rules that are laid out in Table 43. The rules ensure that the first data in a file represent the first time that the time-templating expects to find in that file. Essentially they require that each file holds all possible data for the time period – there cannot be any missing times. Some of these rules are demonstrated in the example section below. If these rules are not followed, the code will detect an error and stop. In Table 43, `dataPerUnits` and `filePerUnits` are the time units that are used to describe the period of the data and the files respectively, chosen from 1 year, 1 month, days, hours and minutes. If a file or data period can be described by more than one time unit, the longer unit is used. For example, a period of 60 minutes is described as 1 hour.
For example, consider daily data held in one file per month. This gives `dataPerUnits`='day' and `filePerUnits`='1 month'. Table 43 shows that the first data in each file must represent the 1^{st} of the month, as might be expected. A file that started with data for the 2^{nd} of the month cannot be used with time templating, even if a particular run does not require the data at that time.

Table 43 Requirements for the time of first data in time templated files.

		dataPerUnits				
		1 year	1 month	days	hours	minutes
filePerUnits	1 year	none	Jan	01Jan	00H 01Jan	00H 01Jan
	1 month	-	none	1^{st} of month	00H 1^{st} of month	00H 1^{st} of month
	days	-	-	none	00H	00H
	hours	-	-	-	none	00H
	minutes	-	-	-	-	none

6.22.2. Variable-name templating

Variable-name templating is so called because it is expected to 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. For example, variable #1 could be in "`file1.dat`", while variable #2 is in "`file2.dat`". Examples of the use of this type of templating are given in the next section. If

[12] Users of GrADS should note that, for these shorter substitution string periods (hours and minutes), JULES can use files that cannot be described by a GrADS template control file. GrADS (at v1.9v4) insists that each file contains data that covers at most one period, whereas JULES allows data for more than one period. For example, if the substitution template includes %h2, GrADS insists that each file contains data for at most one hour, whereas JULES allows each file to have 1, 2, 3, 4..etc hours of data.

using variable name templating with non-SDF formats, the layout of each file must be similar – the number of headers and the number of fields in any time level must be the same in all files.

Table 44 Examples of the use of file name templating.

Substitution template	Description of files	Valid template?	Example file names	Comments
/data/met_data_%y4%mc.dat	Monthly files	Yes	/data/met_data_1990jan.dat /data/met_data_1990feb.dat	
./%y4/met_data_%y4%mc.dat	Monthly files	Yes	./1990/met_data_1990jan.dat	A substitution string can appear more than once. Here data for each year are stored in a separate directory.
%vv_%y4	Yearly files, with each variable in a separate file	Yes	Rain_1990.dat Wind_1990.dat	Variable name and time templating used together. The strings that are to be substituted for %vv will be provided by the user via the run control file.
Data_%d2.dat	Hourly data, each file containing data for 10 days	No		Each file can contain at most 1 day of data. For substitution strings that refer to years, months or days, more than one year/month/day of data can be stored in each file.
Data_%h2.dat	Hourly data, each file containing data for 6 hours.	Yes	Data_00.dat Data_06.dat Data_12.dat Data_18.dat	For substitution strings that refer to hours or minutes, more than one hour or minute of data can be stored in each file.
Data_%mc.dat	Hourly data in monthly files. The time of the first data is 00H 01Jun1990.	Yes	Data_jan.dat Data_feb.dat	
Data_%mc.dat	Hourly data in monthly files. The time of the first data is 01H 01Jun1990.	No	Data_jan.dat Data_feb.dat	Similar to the previous case, but with first data one hour later. In this case, the first data in each file must represent 00H on the 1st of a month. These data cannot be described by a time template and instead the name and time of each data file must be listed (see appropriate section).
Data_%y4.dat	Monthly data in yearly files. The time of the first data is given as 00H	Yes	Data_1990.dat Data_1991.dat	In this case, the time of the first data must be in January. Here it is shown to be a value at approximately mid-month.

	15Jan1990.			

6.23. 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 45. These flags are case-sensitive.

Table 45 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)[13] method that conserves the period means of the driving data file. In order to ensure conservation of the average, these flags can be used only if the data period is an even multiple of the model timestep (i.e., if `driveDataPer=2*n*timestep`; `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.

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

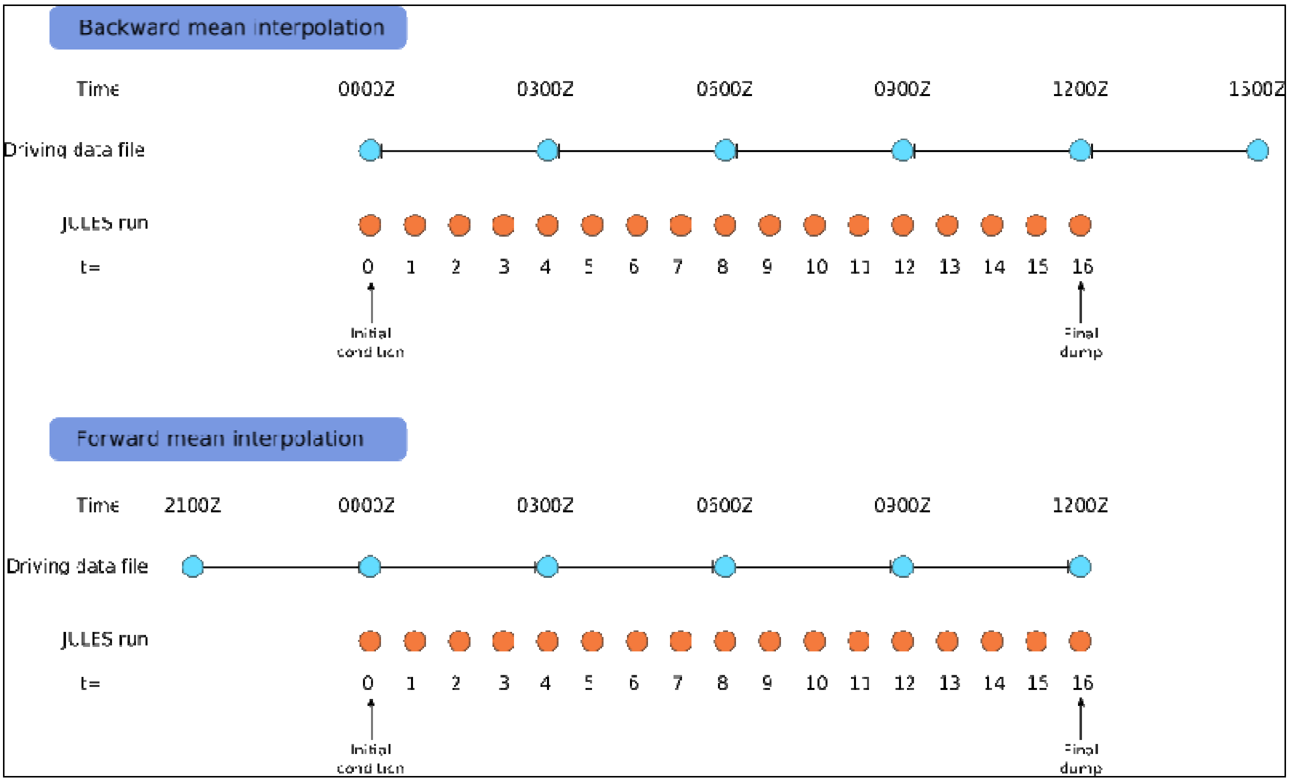

Figure 7. Schematic of JULES interpolation of driving variable from a 3 hour timestep to a 45 minute timestep. Simulation start time is 0000Z (on an arbitrary day) and end time is 1200Z. Blue circles indicate driving data required to complete a JULES simulation from `t=0` to `t=16`. See text for discussion of requirements for driving variables that are forward or backward means.

6.24. Example run control files

Two example run control files come bundled with the JULES source code, in the top-level directory.

`point_loobos_example.jin`
for 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. The results of running this code are also provided in the same directory, so the user can check that their installation of JULES produces results that are acceptably close to those of this standard run.

`point_loobos_triffid_example.jin`
for a single point simulation forced with weather station data. This is similar to `point_loobos_example.jin` above, but with the TRIFFID dynamic vegetation model switched on. No results are provided.

`point_VL92_1T_example.jin, point_VL92_2T_example.jin,`
`point_VL92_M_example.jin`
for a single point run simulation, including the urban land surface types, forced with weather station data. These serve as an example of the original one tile urban scheme, the simple two-tile urban scheme (URBAN-2T) and MORUSES.

`grid_gswp2_example.jin`
for 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 a run control file for a gridded domain.

7. Aspects of the code

7.1. Low-level i/o code

In the course of adding to JULES, a user may well want to read new variables into the model. Most of the input/output of spatial fields is handled by subroutines provided by the module READWRITE_MOD. Particularly important procedures that deal with input are summarised in Table 46. To use this code to read in a new variable, the appropriate procedure should be identified based on the type of variable that is to be read in. For example, to read a field that is only defined on land points, a call to readVar2dComp is appropriate. All these procedures require arguments that define the mapping between the input grid and the model grid.

Note that the choice of procedure is governed solely by the type of variable and is not affected by the shape of the input grid. The correct use of these procedures and the arguments required can be learned by studying the exiting code.

Table 46 Key procedures for reading data.

Name	Summary
`readVar2d`	Reads a variable that is defined at all possible points (both land and sea). The result is a variable on the model grid (this is considered to be a 2-dimensional variable on (x,y), even if the model grid is effectively a vector with ny=1). For example, air temperature is defined at all possible points, both land and sea.
`readVar2dComp`	Reads a variable that is only defined on a subset of points (for example land points). The result is a vector. For example, a land variable can be read from a 2-D (x,y) map (that may contain both land and sea points), and the result is a vector on land points. (The “Comp” in the name is meant to suggest “compression” to a vector!)
`readVar3dComp`	As `readVar2dComp`, but the variable is also a function of the vertical level (e.g. a soil variable on several levels). This 3d version works by looping over the vertical levels, calling the 2d version for each level.

7.2. How to implement new diagnostics for output

The steps needed to add a new diagnostic vary according to what variables are needed in order to calculate the diagnostic. These are covered in the next sections.

7.2.1. Output of existing variables

The data are already held in an existing FORTRAN variable, in a module. This is the easiest case, since the data are easily accessed. The name that is used to select the diagnostic should be added to subroutine `init_out_varlist`, following the existing examples. Care should be taken to specify the correct type of diagnostic (e.g., land points only, soil layers). If the desired diagnostic does not fit any of the existing types, the user may have to closely study the code to work out how to add a new type, and/or contact the JULES developers. Finally, code to load the values into the output space has to be added to subroutine `loadout` (in module `OUTPUT_MOD`). This code may have to calculate the diagnostic using other variables.

7.2.2. Output of new variables

Diagnostics that require variables that the user had added, or that must be calculated in a section of the model code other than the output routines, are more complex to add to JULES. In such a case, it may be easiest to declare a new variable in a FORTRAN module, and to use this variable to hold the values of the diagnostic. Space for the new variable will likely have to be allocated, and the tidiest way to do this would be in the subroutine `allocate_arrays` (which is called at various points during initialisation). The variable can then be accessed by the output procedures and the steps outlined in case 1 above should be followed.

A more sophisticated scheme which only allocated space for a diagnostic if it was required, and loaded the value from any subroutine (avoiding the need to hold the variable in a module, or pass it through the code) has been implemented in some versions of JULES but is not available in the release versions because it is not compatible for use in the Unified Model. If you are keen to get this code, contact the JULES developers.

8. Known limitations of and bugs in the code

1.Limit to longest possible run

The longest possible run that can be attempted with JULES is approximately 100 years. Any 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.

2. Lack of more generic i/o code

If a user wants to introduce new time-varying data that cannot be made to fit into the existing code for vegetation or meteorological data (for example, the new data would need to have the same frequency as the other data type), they may have a substantial job on their hands! For many purposes, a simple 'hack' may suffice (e.g. write code to read a particular data set for a particular run), but this will lack generality and options such as automatic spin up will be hard to accommodate. At present there is no good solution – we don't have any flexible coupling code that can be told to fetch suitable values of an arbitrary field, although JULES may move towards this in future.

3. Spin up over short periods

The current code 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.

9. Variables available for output

Variables that are available for output from JULES are listed in the tables of this section, separated according to their type. Types of variables are:

SINGLE: a single value at all gridpoints (land and sea) (Table 47).
LAND: a single value at land gridpoints (Table 48).
PFT: a value for each of `npft` PFTs at each land gridpoint (Table 49).
TILE: a value for each of `ntiles` tiles at each land gridpoint (Table 50).
TYPE: a value for each of `ntype` surface types at each land gridpoint (Table 51).
SOIL: a value for each of `sm_levels` soil layers at each land gridpoint (Table 52).
SNOW: a value for each of nsmax snow layers at each tile at each land gridpoint (Table 53).
SC: a value for each of *N* soil carbon pools at each land gridpoint (Table 54). *N*=1 if `l_triffid`=FALSE, else *N*=4.

These tables were correct at the time of writing, but the most reliable way to determine exactly which variables are available for a particular version of JULES is to look at the variables listed in the subroutine `init_out_varlist`, and which can be echoed to screen at the start of a JULES run by setting `echo=TRUE` (see Section 6.2).

A few variables are not available in the standard release (for reasons of compatibility with the Unified Model - see Section 7.2.2), but can be accessed with the addition of extra code which can be requested from the JULES office. These “offline” variables are shown in italics in the tables below.

Table 47 A list of output variables that have a single value at each gridpoint.

Name	Description
conRain	Gridbox convective rainfall (kg m^{-2} s^{-1})
conSnow	Gridbox convective snowfall(kg m^{-2} s^{-1})
cosz	Cosine of the zenith angle (-)
diffFrac	Gridbox fraction of radiation that is diffuse (-)
ecan	Gridbox mean evaporation from canopy/surface store (kg m^{-2} s^{-1})
ei	Gridbox sublimation from lying snow or sea-ice (kg m^{-2} s^{-1})
esoil	Gridbox surface evapotranspiration from soil moisture store (kg m^{-2} s^{-1})
fqw	Gridbox moisture flux from surface (kg m^{-2} s^{-1})
ftl	Gridbox surface sensible heat flux (W m^{-2})
landAlbedo1	Gridbox albedo for waveband 1 (direct beam visible)
landAlbedo2	Gridbox albedo for waveband 2 (diffuse visible)
landAlbedo3	Gridbox albedo for waveband 3 (direct beam NIR)
landAlbedo4	Gridbox albedo for waveband 4 (diffuse NIR)
latentHeat	Gridbox surface latent heat flux (W m^{-2})
latitude	Gridbox latitude (°)
longitude	Gridbox longitude (°)
lsRain	Gridbox large-scale rainfall (kg m^{-2} s^{-1})
lsSnow	Gridbox large-scale snowfall (kg m^{-2} s^{-1})
LWdown	Gridbox surface downward LW radiation (W m^{-2})
precip	Gridbox precipitation rate (kg m^{-2} s^{-1})
pstar	Gridbox surface pressure (Pa)
q1p5m	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})
snomltSurfHtf	Gridbox heat flux used for surface melting of snow (W m^{-2})
snowfall	Gridbox snowfall rate (kg m^{-2} s^{-1})
snowMass	Gridbox snowmass (kg m^{-2})
surfHtFlux	Gridbox net downward heat flux at surface over land and sea-ice fraction of gridbox (W m^{-2})
SWdown	Gridbox surface downward SW radiation (W m^{-2})
t1p5m	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	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})

Table 48 A list of output variables that have a single value at each land gridpoint.

Name	Description
albedoLand	Gridbox albedo (as used to calculate net shortwave radiation) (-)
canopy	Gridbox canopy water content (kg m^{-2})
cs	Gridbox total soil carbon (kg C m^{-2})
cv	Gridbox mean vegetation carbon (kg C m^{-2})
depthFrozen	Gridbox depth of frozen ground at surface (m)
depthUnfrozen	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	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	Gridbox soil moisture availability factor (beta) (-)
fwetl	Gridbox wetland fraction (-)
gpp	Gridbox gross primary productivity (kg C $m^{-2}s^{-1}$)
gs	Gridbox surface conductance to evaporation (m s^{-1})
hfSnowMelt	Gridbox snowmelt heat flux (W m^{-2})
landIndex	Index (gridbox number) of land points
liceIndex	Index (gridbox number) of land ice points
litCMn	Gridbox mean carbon litter (kg C m^{-2} $(360\text{days})^{-1}$)
LWnet	Gridbox surface net LW radiation (W m^{-2})
LWup	Gridbox surface upward LW radiation (W m^{-2})
npp	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})
radnet	Surface net radiation (W m^{-2})
respP	Gridbox plant respiration (kg C m^{-2} s^{-1})
respS	Gridbox total soil respiration (kg C m^{-2} s^{-1})
respSDrOut	Gridbox mean soil respiration for driving TRIFFID (kg C m^{-2} $(360\text{days})^{-1}$)
runoff	Gridbox runoff rate (kg m^{-2} s^{-1})
sat_excess_roff	Gridbox saturation excess runoff rate (kg m^{-2} s^{-1})
smcAvailTop	Gridbox available moisture in surface layer of depth given by zsmc (kg m^{-2})
smcAvailTot	Gridbox available moisture in soil column (kg m^{-2})
smcTot	Gridbox total soil moisture in column (kg m^{-2})
snomltSubHtf	Grdbox sub-canopy snowmelt heat flux (W m^{-2})
snowCan	Gridbox snow on canopy (kg m^{-2})
snowDepth	Gridbox depth of snow (m)
snowFrac	Gridbox snow-covered fraction of land points (-)
snowFracAlb	Gridbox average weight given to snow for albedo (-)
snowGrCan	Gridbox average snow beneath canopy (snow_grnd) (kg m^{-2})
snowIceTot	Gridbox frozen water in snowpack (kg m^{-2}) Only available if nsmax>0.
snowLiqTot	Gridbox liquid water in snowpack (kg m^{-2})

	Only available if `nsmax`>0.
`snowMelt`	Gridbox rate of snowmelt (kg m^{-2} s^{-1})
`soilIndex`	Index (gridbox number) of soil points
`sthZw`	Sol wetness in the deep (water table) layer (-)
`subSurfRoff`	Gridbox sub-surface runoff (kg m^{-2} s^{-1})
`surfRoff`	Gridbox surface runoff (kg m^{-2} s^{-1})
`surfRoffInf`	Gridbox infiltration excess surface runoff (kg m^{-2} s^{-1})
`swetLiqTot`	Gridbox unfrozen soil moisture as fraction of saturation (-)
`swetTot`	Gridbox soil moisture as fraction of saturation (-)
`SWnet`	Gribox net shortwave radiation at the surface (W m^{-2})
`tfall`	Gridbox throughfall (kg m^{-2} s^{-1})
`trad`	Gridbox effective radiative temperature (K)
`wFluxSfc`	Gridbox downwards moisture flux at soil surface (kg m^{-2} s^{-1})
`zw`	Gridbox depth to water table (m)

Table 49 A list of output variables that have a single value for each PFT at each land gridpoint.

Name	Description
`cVegP`	PFT total carbon content of the vegetation (kg C m^{-2})
`canhtP`	PFT canopy height (m)
`ciP`	PFT internal CO2 pressure (Pa)
`fluxO3Stom`	PFT flux of O3 to stomata (mol m-2 s-1)
`fsmcP`	PFT soil moisture availability factor (-)
`gLeafP`	PFT leaf turnover rate ($[360days]^{-1}$)
`gLeafDayP`	PFT mean leaf turnover rate for input to PHENOL ($[360days]^{-1}$)
`gLeafDrOutP`	PFT mean leaf turnover rate for driving TRIFFID ($[360days]^{-1}$)
`gLeafPhenP`	PFT mean leaf turnover rate over phenology period($[360days]^{-1}$)
`gstomP`	PFT bulk (canopy) stomatal conductance for water vapour (m s^{-1})
`gppP`	PFT gross primary productivity (kg C m^{-2} s^{-1})
`laiP`	PFT leaf area index (-)
`laiPhenP`	PFT leaf area index after phenology (-)
`litCP`	PFT carbon litter (kg C m^{-2} $(360days)^{-1}$)
`nppDrOutP`	PFT mean NPP for driving TRIFFID (kg C m^{-2} $(360days)^{-1}$)
`nppP`	PFT net primary productivity (kg C m^{-2} s^{-1})
`o3ExpFac`	PFT ozone exposure factor
`rdcP`	Canopy dark respiration, without soil water dependence (mol CO_2 m^2 s^{-1})
`respPP`	PFT plant respiration (kg C m^{-2} s^{-1})
`respWDrOutP`	PFT mean wood respiration for driving TRIFFID (kg C m^{-2} $(360days)^{-1}$)
`respWP`	PFT wood respiration (kg C m^{-2} s^{-1})

Table 50 A list of output variables that have a single value for each tile at each land gridpoint.

Name	Desciption
`alb1T`	Tile land albedo, waveband 1 (direct beam visible)
`alb2T`	Tile land albedo, waveband 2 (diffuse visible)
`alb3T`	Tile land albedo, waveband 3 (direct beam NIR)
`alb4T`	Tile land albedo, waveband 4 (diffuse NIR)
`anthropHtFluxT`	Anthropogenic heat flux for each tile (W m^{-2})
`canopyT`	Tile surface/canopy water for snow-free land tiles (kg m^{-2})
`catchT`	Tile surface/canopy water capacity of snow-free land tiles (kg m^{-2})
`ecanT`	Tile evaporation from canopy/surface store for snow-free land tiles (kg m^{-2} s^{-1})
`eiT`	Tile sublimation from lying snow for land tiles (kg m^{-2} s^{-1})
`emisT`	Tile emissivity
`esoilT`	Tile surface evapotranspiration from soil moisture store for snow-free land tile (kg m^{-2} s^{-1})
`fqwT`	Tile surface moisture flux for land tiles (kg m^{-2} s^{-1})
`ftlT`	Tile surface sensible heat flux for land tiles (W m^{-2})
`gcT`	Tile surface conductance to evaporation for land tiles(m s^{-1})
`leT`	Tile surface latent heat flux for land tiles (W m^{-2})
`nsnow`	Tile number of snow layers (-)
`q1p5mT`	Tile specific humidity at 1.5m over land tiles (kg kg^{-1})
`radnetT`	Tile surface net radiation (W m^{-2})
`rgrainT`	Tile snow surface grain size (μm)
`snowCanMeltT`	Tile melt of snow on canopy (kg m^{-2} s^{-1})
`snowCanT`	Tile snow on canopy (kg m^{-2})
`snowDepthT`	Tile snow depth (m)
`snowGrCanMeltT`	Tile melt of snow under canopy (kg m^{-2} s^{-1})
`snowGroundRhoT`	Tile bulk density of snow on ground (kg m^{-3})
`snowGrCanT`	Tile snow on ground below canopy (kg m^{-2})
`snowGroundT`	Tile snow on ground (snow_tile or snow_grnd) (kg m^{-2})
`snowIceT`	Tile total frozen mass in snow on ground (kg m^{-2}) Only available if `nsmax`>0.
`snowLiqT`	Tile total liquid mass in snow on ground (kg m^{-2}) Only available if `nsmax`>0.
`snowMassT`	Tile lying snow (total) (kg m^{-2})
`snowMeltT`	Tile snow melt rate (melt_tile) (kg m^{-2} s^{-1})
`surfHtFluxT`	Downward heat flux for each tile (W m^{-2})
`surfHtStoreT`	C*(dT/dt) for each tile (W m^{-2})
`t1p5mT`	Tile temperature at 1.5m over land tiles (K)
`tstarT`	Tile surface temperature (K)
`z0T`	Tile surface roughness (m)

Table 51 A 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.
`tileIndex`	Index (gridbox number) of land points with each surface type

Table 52 A list of output variables that have a single value for each soil level at each land gridpoint.

Name	Description
`bSoil`	Brooks-Corey exponent for each soil layer (-)
`ext`	Extraction of water from each soil layer (kg m^{-2} s^{-1})
`hCapSoil`	Soil heat capacity (J K^{-1} m^{-3}) for each soil layer
`hConSoil`	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})
`soilWet`	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 (-)
`tSoil`	Sub-surface temperature of each layer (K)
`vsmcCrit`	Volumetric moisture content at critical point for each soil layer (-)
`vsmcSat`	Volumetric moisture content at saturation for each soil layer (-)
`vsmcWilt`	Volumetric moisture content at wilting point for each soil layer (-)
`wFlux`	Downwards moisture flux at bottom of each soil layer (kg m^{-2} s^{-1})

Table 53 A list of output variables that have a single value for each snow layer at tile each land gridpoint.

Name	Description
`rGrainL`	Grain size in snow layers for each tile (μm)
`snowDs`	Depth of each snow layer for each tile (m)
`snowIce`	Mass of ice in each snow layer for each tile (kg m^{-2})
`snowLiq`	Mass of liquid water in each snow layer for each tile (kg m^{-2})
`tsnow`	Temperature of each snow layer (K)

Table 54 A list of output variables that have a single value for each soil carbon pool at each land gridpoint.

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

10. List of Tables