Joint UK Land Environment Simulator (JULES) User Guide

Release 7.3

Met Office

Sep 12, 2023

CONTENTS

The Joint UK Land Environment Simulator (JULES) is a computer model that simulates many soil and vegetation processes. This guide 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 JULES v2.1 these differences were reconciled with the UM. As of JULES v3.1, a single code repository is used for both standalone JULES and JULES in the UM.

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

CHAPTER ONE

RELEASE NOTES

1.1 JULES version 7.3 Release Notes

The JULES vn7.3 release consists of approximately 19 tickets from 9 authors, including work by many other people. Full details of the tickets committed for JULES vn7.3 can be found on the JULES shared repository Trac system. Ticket numbers are indicated below, e.g. #1180.

1.1.1 General/Technical changes

- Removed default numerical values from variables in the *JULES_HYDROLOGY* and *JULES_RIVERS* namelists. (#1180)
- Allow a model domain that straddles the edge of the input grid (for grids that are cyclic in longitude). (#1301)
- Simplification of the namelists required for OASIS-Rivers to aid maintenance of the LFRic coupled miniapp. (#1385)
- Water resource variables bundled in a new TYPE. (#1401)
- Clarification of timestep-related variables. (#1403)
- Change kind type names to avoid clash with Fortran intrinsics. (#1408)
- Removed some include files by moving code to module files. (#1411, 1412. 1418)

1.1.2 Bugs fixed

- Fixed standalone diagnostics *sat_excess_roff* and *drain*. (#1039)
- Fix for possible floating point exceptions in veg2 code. (#1155)
- Bug fixes for irrigation code. (#1386)
- Bug fixes for OASIS-Rivers coupling field, *rflow_outflow*. (#1435)

1.1.3 Changes to testing

- Updated *suite_report.py*. (#1442)

1.1.4 Documentation updates

- Simplified the description of platform files. (#1353)
- Minor syntax changes in the JULES Coding Standards. (#1402)
- Minor corrections to code and documentation. (#1421, 1443)
- Updates associated with many of the above changes, and release notes. (#1384)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.2 JULES version 7.2 Release Notes

The JULES vn7.2 release consists of approximately 19 tickets from 17 authors, including work by many other people.

Full details of the tickets committed for JULES vn7.2 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #1317.

1.2.1 General/Technical changes

- Switch *l_coord_latlon* specifies if the coordinate system is latitude-longitude or something else (e.g. a rotated grid). This makes it easier to model and postprocess grids that are not defined by latitude and longitude. (#1317)
- Addition of OASIS coupling capabilities to the river routing executable. (#1191)
- The RothC soil C model is now referred to as the 4-pool model. (#1348)
- Removed the remaining 2D field being passed into JULES via a module, replacing it with an argument list variable. (#1376)
- Removed a redundant print statement from the urban code. (#1367)
- Updated CABLE (as part of consolidating CABLE across its main applications, ACCESS-CM2, ESM1.5 and CABLE standalone), also corrected an inconsistency in the order of arguments for a few subroutines. (#1373)
- Increase compile-time checking with Cray's CCE. (#1061)
- Updated configuration for NIWA. (#1387)

1.2.2 Bugs fixed

- Improved numerical implementation of an EXP calculation in the spectral albedo scheme. (#1189)
- Full initialisation of soil carbon arrays used with the INFERNO fire model. (#1356)
- Fixed the upgrde macro chain. (#1368)
- Fixes to umdp3_checker. (#1397)

1.2.3 Changes to testing

- Rose stem testing extended to include overbank inundation. (#999)
- loobos_gl4_cable now generates output files and is included in rose stem testing at several sites. (#1375)
- Updated suite_report.py and changed rosestem_branch_checker.py to use generic variable names to match the UKCA version. (#1369, 1379)
- Refactored rose stem tests for building JULES with FAB. (#1354)

1.2.4 Documentation updates

- Updates associated with many of the above changes, and release notes. (#1381)
- Updated JULES docs to ensure that they build with a more recent version of Sphinx - they can now be built using Sphinx 2.4.0 and Python 3.6.10. (#1382)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.3 JULES version 7.1 Release Notes

The JULES vn7.1 release consists of approximately 20 tickets from 18 authors, including work by many other people.

Full details of the tickets committed for JULES vn7.1 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #931.

1.3.1 Science changes

- Interactive gas-phase deposition routines from UKCA have been added to JULES, together with a variant version that removes the restriction on the surface tile configuration. See `dry_dep_model`. (#931)

1.3.2 General/Technical changes

- Added a switch to allow interpretation of times in the model and the driving data as local solar time - see `l_local_solar_time`. (#1327)
- Added a switch `l_drive_with_global_temps` so that JULES can be driven with global temperatures and climate patterns. (#1322)
- Made Medlyn stomatal conductance, Farquhar C_3 photosynthesis, and thermal acclimation available in the UM. (#1246)
- Further work towards allowing layered soil carbon (`l_layeredc` = TRUE) in the UM. (#1237)
- A varying grey tile emissivity has been passed in to JULES - currently only available if selected in LFRic. (#1247)
- OMP improvements in various routines. (#1310)
- Enabled a call to CABLE albedo (part of ongoing work to provide access to CABLE). (#1314)
- MORUSES and anthropogenic heat related metadata migrated to jules-shared to facilitate their implementation in LFRic. The simple two-tile urban scheme now no longer requires W/R as an input unless the total urban fraction only is specified in the ancillary data. (#1255)
- Included the fab_jules app which builds JULES using Fab software. (#1331, 1364)
- Updated compiler in JASMIN gfortran platform file. (#1347)
- Upgraded rose stem testing on Met Office EXZ to use latest availiable CPE, and fixes for EXZ. (#1362, 1358)

1.3.3 Bugs fixed

- Corrected a bug in the *surf_ht_flux* diagnostic when using the Flake lake model. (#1340)
- Fixing problems in the WARNING output messages. (#1303)
- Fixed a metadata/rose config error introduced by a bug in Rose. (#1363)

1.3.4 Changes to testing

- Updated NCI rose stem tests. (#1266)

1.3.5 Documentation updates

- Updates associated with many of the above changes and to module leaders file, and release notes. (#1275, 1343)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.4 JULES version 7.0 Release Notes

The JULES vn7.0 release consists of approximately 31 tickets from 20 authors, including work by many other people.

Full details of the tickets committed for JULES vn7.0 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #911.

1.4.1 Science changes

- New functionality for modelling bioenergy crops - see `l_trif_biocrop` and `l_ag_expand`. (#911)
- Implemented a socio-economic factor in the fire ignition and suppression parameterisation in INFERNO, based on a Human Development Index (HDI). (#1284)
- A new logical switch `l_mo_buoyancy_calc` enables an interactive buoyancy in the calculation of the surface transfer coefficients. (#1242)

1.4.2 General/Technical changes

- Surface type IDs fully extended to JULES to allow extra surface configuration checks to take place. A routine to check compatible science options is now accessible to all parent models, allowing cross-namelist checking to take place once all the namelists have been read, removing the dependency on order. (#1249)
- Upgraded FLake driver to version 1.10 to include a bug fix from the FLake community and keep our copy of FLake aligned with the community code base. (#1227)
- JULES now passes sea ice surface heat flux (surf_ht_flux_sice), sea ice top melt (sice_melt) and sea ice sublimation (ei_sice) from JULES to LFRic as part of the fluxes structure. These variables, and a few others, are no longer weighted by sea ice fraction before being passed out of JULES. The weighting by sea ice fraction should be done in the parent models. (#1259)
- Made stencil used in buddy_sea option work for unstructured meshes, rather than assuming i+1, j+1 indexing will work. (#1286)
- Tidied soil code so that arguments follow UMDP3 order, and corrected some argument INTENTs. (#843)
- Streamlined standalone code dealing with input of ancillary fields. (#1256)
- In preparation for including layered soil carbon in the UM, the soil respiration returned to the UM now has an extra dimension which can potentially be used to represent layers. (#1236)

- Migration of *JULES_NVEGPARM* and *JULES_SURFACE_TYPES* used in LFRic to shared metadata held in jules-shared. Checking of *JULES_NVEGPARM* moved to a new shared routine and added to UM. (#1272)
- Passing CABLE vars (TYPEs) from top level through to and into surf_couple layer. (#1226)
- Updated the module load of the make_jules_release script. (#1263)

1.4.3 Bugs fixed

- Fix for precision issue whereby infiltration of rainfall into snowpack could become larger than the incident rainfall, resulting in negative large-scale rain. (#1092)
- Bug fix for persistent small snow amounts - see *l_fix_snow_frac*. (#1279)
- Correction to the chlorophyll dependence of the oceanic albedo in the scheme of Jin et al. (2011). (#1260)
- Bug fix to allow calculation of the lw_net diagnostic. (#1270)
- Fixes to UM routines identified by the NAG compiler. (#1276)
- Corrected namelist reading to ensure that FLake can be run with urban2t and MORUSES schemes in standalone JULES. (#1277)
- Fixed bug that prevented finalisation of initial output files. (#1281)
- Introduced namelist variable *coordinate_file* to fix bug preventing use of file-name templating with river ancillaries. (#1287)
- Prevent faults caused by attempting to read an absent dummy argument. (#1292)
- Fixed benign OMP bug in snowpack_mod. (#1304)

1.4.4 Changes to testing

- Added rose stem test for the FLake lake model. (#1277)
- Altered the eraint rose stem apps to better represent river routing. (#1299)
- Increased resources requested for build in Met Office XC40 rose stem. (#1291)
- Changes to module load, mpiexec usage, and memory requested for rose stem for the Met Office EXZ platform. (#1296, 1302, 1305, 1309)

1.4.5 Documentation updates

- Updates associated with many of the above changes and to module leaders file, and release notes. (#1275, 1300)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.5 JULES version 6.3 Release Notes

The JULES vn6.3 release consists of approximately 28 tickets from 14 authors, including work by many other people.

Full details of the tickets committed for JULES vn6.3 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #1142.

1.5.1 Science changes

- Restructured the parameterisation of thermal acclimation of photosynthesis to permit a wider range of configurations - see `photo_acclim_model`. (#1142)
- Updated soil N limitation in layered CN model (only applies when `l_layeredc` = TRUE). (#1213)

1.5.2 General/Technical changes

- Improved nitrogen leaching for vertically-resolved soil biogeochemistry. (#1219)
- Allowed the use of daily files for input and output (via time templating for input files). (#1215)
- Further developments towards the RED vegetation model. (#1182)
- Refactored IMOGEN in preparation for adding netcdf reading and variable resolution. (#1214)
- Added functionality for the first phase of a standalone Rivers executable. This uses JULES I/O and is still under development; it currently is not scientifically correct as a result of a timestamp issue and does not include dumping capability. (#1084)
- Namelists *jules_urban2t_param* and *jules_urban_switches* amalgamated into `JULES_URBAN`, which is also now consistent with the module name. (#1077)
- Updated metadata for bedrock and layered soil carbon. (#1234)
- Framework for a shared metadata solution introduced initially with namelist items from *jules_surface* and *jules_vegetation* currently shared with LFRic. (#1195) More information on sharing JULES metadata can be found here: SharingJULESmetadata
- Minor adjustment to how FLake variables are dealt with during initialisation. (#1169)
- Modularised and refactored subroutine vgrav. (#1199)
- Rivers variables bundled in a new TYPE. (#1176)
- Modularised further UM-only files. (#1221)
- Removed the soil carbon variable cs from the UM, in prepration for the introduction of layered soil carbon. (#1210)
- Removed snowmelt_ij from fluxes_mod. (#1167)
- Tidied code to remove compiler warnings. (#1245)
- Revised the implementation of defining the veg/soil parameters to be passed to CABLE as a single TYPE, following them being read in through namelists. We also implement a working variables TYPE to hold variables at the top-level so that they may be passed between pathways (i.e explicit, implicit); as well as between time steps (at least until they are elevated to prognostics level). (#1223)
- Added a further restriction on the type of processor used for rose stem testing on JASMIN. (#1222)
- Updated list of module leaders. (#1240)

1.5.3 Bugs fixed

- Fixes to FLake surface exchange and decoupling FLake from the soil column. (#1094)
- Added missing metadata for PFT parameter *dust_veg* (only used in the UM). (#1206)
- Corrected the definitions of the surface longwave radiation diagnostics. (#1220)
- Fixed a bug that prevented files from being closed when running with MPI. (#1241)
- Ensured that soilеccose types are set up correctly. (#1229)
- Ensured Python3 is in path on both MONSooN and JASMIN systems. (#1216, 1231)

1.5.4 Documentation updates

- Updates associated with many of the above changes, and release notes. (#1243)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.6 JULES version 6.2 Release Notes

The JULES vn6.2 release consists of approximately 23 tickets from 15 authors, including work by many other people.

Full details of the tickets committed for JULES vn6.2 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #470.

1.6.1 Science changes

- Added the ability to label and trace a subset of the soil carbon in the RothC layered model - see `l_label_frac_cs`. (#470)
- Replaced original logical for corrections to coastal tiling with a 3 way switch (`ctile_orog_fix`). The latest improved fix combines the previous fix over sea with the original behaviour over land. (This is mainly relevant for runs with the Unified Model.) (#1184)

1.6.2 General/Technical changes

- Improvements and bug fixes for river grid and flow directions. (#1170)
- Selected output variables are now available on the river grid. (#1163)
- Technical work coupling the RED vegetation model to JULES. This is early in a staged process and RED is not yet available for general use. (#1034)
- Only check for duplicate lat-lon coordinates if those are set using constant values. (#1164)
- Variables related to fluxes and atmospheric chemistry bundled into TYPEs. (#1104, 1159)
- Removed momentum calculations for LFRic as those are now done in bespoke LFRic code. (#1144)
- Further work towards enabling interoperability between the JULES and CABLE land surface models - this step dealing with CABLE prognostic variables (see `CABLE_PROGS`). (#1131)
- Removed most compiler warnings flagged by UM builds. (#1187)
- Ported the JULES codebase to the Met Office's EX1A system. (#1193)
- Updated Python scripts to be Python 3 compatible. (#1091)
- Preparing the JULES vn6.2 release. (#1204)

1.6.3 Bugs fixed

- Bug fix to correct an array addressing issue in snow albedo calculations with `l_embedded_snow` = TRUE. (#1064)
- Deallocating a few more arrays in ancil_info. (#1190)
- Fix to ensure that in coupled NWP models lake ice temperatures (where lakes are defined as sea points) evolve correctly - this is fixed by setting `l_fix_lake_ice_temperatures` = TRUE. (#1161)
- Prevent error in internal write of ctile_orog_fix (potentially many characters long) into a 3 character buffer. (#1205)
- Fix arguments to subroutine next_time. (#1209)

1.6.4 Documentation updates

- Updates associated with many of the above changes, and release notes. (#1156, 1186, 1192, 1198)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.7 JULES version 6.1 Release Notes

The JULES vn6.1 release consists of approximately 21 tickets from 13 authors, including work by many other people.

Full details of the tickets committed for JULES vn6.1 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #949.

1.7.1 General/Technical changes

- Added the technical infrastructure to activate irrigation in the UM (atmosphere model). (#949)
- Irrigation timestep is controlled by `nstep_irrig`. (#1146)
- Minor restructuring of water resources code. (#1122)
- The handling of the MORUSES roughness length in the surface exchange has been made more robust and the functionality for calculating urban morphology from the empirical relationships of Bohnenstengel et al. 2011 has also been fixed. (#1106)
- Forcing and lake variables bundled into TYPEs. (#1136, 1135)
- Conversion of initialisation and io/dump include files to modules, including some reorganisation required for River.Exe (standalone rivers-only executable). (#1158)
- CABLE lai_pft and canht_pft initialised from new namelist `CABLE_PFTPARM`. (#1119)
- Miscellaneous small corrections to code and documentation. (#1143)
- Added new 1.5m visibility diagnostics to sfdiags (only for UM). (#1134)
- Improved performance in RA3 configurations. (#1113)
- Added further platform files for JASMIN. (#1145)

1.7.2 Bugs fixed

- Corrected previously uninitialised PFT parameters when coupled to the UM. (#1137)
- Minor bug fixes and metadata updates for IMOGEN. (#1126)
- Bug fixes for seed_rain. (#1129, 1166)
- Prevent memory issues when using the INFERNO fire scheme in UM configurations. (#1149)
- Bug fix to add Rose metadata and upgrade macro for `JULES_VEGETATION_PROPS` namelist. (#1141)

1.7.3 Changes to testing

- Reflected changes in the JASMIN environment (parallel netcdf and SLURM scheduler) in the settings for rose stem. (#1120)

1.7.4 Documentation updates

- Revised presentation of the available diagnostics. (#1139)
- Updates associated with many of the above changes, and release notes. (#1157)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.8 JULES version 6.0 Release Notes

The JULES vn6.0 release consists of approximately 18 tickets from 10 authors, including work by many other people.

Full details of the tickets committed for JULES vn6.0 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #847.

1.8.1 Science changes

- Added a microbial scheme for methane production (see *l_ch4_microbe*). (#847)
- Changes to soil evaporation: added a PFT-specific factor to multiply soil conductance under the canopy (*gsoil_f_io*) and a switch to limit soil conductance above the critical point (*l_limit_gsoil*). (#1093)
- Added option (l_icerough_prognostic) to use the prognostic sea ice roughness length calculated in CICE, rather than a constant value (available only when coupled). (#583)
- Calculate sea ice penetrating solar radiation to be sent to sea ice model. (#1100)

1.8.2 General/Technical changes

- Irrigation code adapted for use with *l_water_resources* = TRUE. (#1086)
- Surface type IDs have been made available to JULES on a switch `l_surface_type_ids`, however they currently have no functionality and are not passed to the JULES I/O. As part of the metadata consolidation project the namelists *JULES_SURFACE_TYPES* and *JULES_LSM_SWITCH* are now consolidated, the latter by amalgamating it with *JULES_MODEL_ENVIRONMENT*. The GUI panels are also now consistently named in UM and standalone. In parallel runs, informative output can now also be limited to Task 0 (see *JULES_PRNT_CONTROL*). (#1095)
- Prevented duplicate (lat,lon) pairs being prescribed. (#1107)
- Extended the TYPE design to sweep up several small modules. (#1102)
- Optimisation in snow control routine. (#1110)
- Removed some default values for values read via the *JULES_DRIVE* namelist, and improved triggering. (#1033)
- Initialised further namelist variables before use. (#598)
- Fixed bad characters in metadata. (#1103)
- Further work to keep JULES and UM metadata consistent. (#1118)

1.8.3 Bugs fixed

- Added option (*l_accurate_rho*) to use accurate calculation of surface air density in surface exchange calculations. (#194)
- Fix to remove erroneously large river flows at river mouths with RFM (*i_river_vn* = 2). (#1081)
- Fixed unallocated TRIFFID variable in the new progs structure. (#1109)

1.8.4 Documentation updates

- Updates associated with many of the above changes, and release notes. (#1097, 1112)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.9 JULES version 5.9 Release Notes

The JULES vn5.9 release consists of approximately 21 tickets from 11 authors, including work by many other people.

Full details of the tickets committed for JULES vn5.9 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #499.

1.9.1 Science changes

- The FLake lake model is now available in standalone JULES (as well as in the UM), modelling sub-surface conditions on the lake tile - see *l_flake_model*. (#499)
- Fix to stop snow from accumulating on unfrozen lakes when running the FLake lake model. (#1057)
- Options to disaggregate the albedo of bare soil between the VIS and NIR regions and to include the zenith-angle dependence of the bare soil albedo have been included. See *l_hapke_soil* and *l_partition_albsoil*. (#1020)

1.9.2 General/Technical changes

- Modifications to ensure the irrigation namelist for standalone JULES is consistent with the UM namelist structure. This involved moving irrigation switches to namelist *JULES_IRRIG* and adding the *JULES_IRRIG_PROPS* namelist for ancillary fields. (#838)
- Added initial control-level code for water resources. (#1018)
- The interface for the *river_control* subroutine now follows the model used by the *surf_couple_** routines. This is part of activities to create a standalone river routing capability. (#1066)
- Added latent heat flux diagnostics separately for land and sea/sea-ice (Unified Model runs only). (#992)
- Surface diagnostics for 1.5m temperature, specific and relative humidity, dewpoint temperature, and fog fraction separately over ocean (sea + sea ice) and land made available in the UM (not available in standalone JULES). (#1074)
- Removed unused arguments, to avoid UM compiler warnings. (#1069)
- Optimisation of code used in GA9. (#1071)
- Added *ONLY*s to all *USE* statements that did not have them, to avoid undesirable effects. (#1072)
- All transparent, non-functional metadata consolidated with the UM, including *sort-key*, *description*, *url* and *help*, which has been removed where duplicated by the JULES user guide or merged in. (#1055)
- Keep standalone and UM JULES meta data consistent. (#1083)

1.9.3 Bugs fixed

- Fixed bug that affected the reading of soil ancillary fields when *const_z* = TRUE in the JULES_SOIL_PROPS namelist. The bug had the potential to result in some soil ancillary fields being zero - which would likely have resulted in an obviously wrong result and/or a model crash. (#1080)
- Fixed bug in the reading of dumps containing rain_seed fields. (#1059)
- Fixes for various issues related to argument intents. (#1058, 1062)
- Fix for UM configs with iseasurfalg=2 (m10 now initialised). (#975)

1.9.4 Changes to testing

- The gswp2_irrig_limit rose stem tests are included in the set run at UKCEH. (#1053)
- Added a further rose stem test with irrigation. (#838)

1.9.5 Documentation updates

- Updates associated with many of the above changes, and release notes. (#1078)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.10 JULES version 5.8 Release Notes

The JULES vn5.8 release consists of approximately 19 tickets from 9 authors, including work by many other people. This was an unusual release cycle as it was primarily aimed at technical developments in support of changes to the Unified Model system.

Full details of the tickets committed for JULES vn5.8 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #919.

1.10.1 General/Technical changes

- The surf_couple* routines have been restructured to separate land and sea/sea-ice parts - making the code clearer and facilitating the implementation of CABLE into the JULES framework. (#919)
- Fields passed by argument rather than via USE statements. (#1022, #1024, #1025, #1028, #1029, #1030, #1037, #1038, #1043)
- Added *dump_period_unit* to allow the units of *dump_period* to be years or seconds. (#1021)
- Removal of the old qsat routines. (#1015)
- Access the gather_field and scatter_field subroutines via modules. (#1051)
- The url fields in HEAD metadata now have the “latest” tag, allowing both the UM and JULES to have the same url field. (#1017)

1.10.2 Bugs fixed

- Bug fix to allow runs with soil tiling that do not rely on broadcastng values. (#1048)
- Bug fix to prevent array bounds error with river routing, and other minor changes to river routing. (#1049)
- Fix to allow compilation with pgfortran as part of UM-JULES. (#1010)

1.10.3 Changes to testing

- Tidied some rose stem apps. (#1050)

1.10.4 Documentation updates

- Updates associated with many of the above changes, and release notes. (#1040)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.11 JULES version 5.7 Release Notes

The JULES vn5.7 release consists of approximately 34 tickets from 16 authors, including work by many other people.

Full details of the tickets committed for JULES vn5.7 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #548.

1.11.1 Science changes

- Soil tiling made available - see `l_tile_soil`. Each land gridbox can be modelled using a single soil column or with a separate soil column for each surface tile. (#548)
- Enabled the simulation of multiple crops in a growing season, and therefore crop rotations - see `l_croprotate`. (#821)
- Added thermal adaptation and acclimation of leaf-level photosynthesis - see `photo_acclim_model`. (#863)
- Remaining source/sink terms for inorganic nitrogen included with the ECOSSE soil model (`soil_bgc_model` = 3). (#788)

1.11.2 General/Technical changes

- New functionality added to allow for a new vegetation biogeochemical model. Introduced more extensive use of modules to control the read and write of variable states, and FORTRAN type objects to control data flows. This code is not yet suitable for general use. (#888)
- Technical work for the implementation of the Robust Ecosystem Demography (RED) dynamic vegetation model. (#902)
- Moved allocation statements into the modules that hold the variables (and out of the monolithic allocate_jules_arays.F90). (#978)
- Improved processing of the values read from namelist `JULES_OVERBANK`. (#987)
- Added a KIND type to the declarations of REAL variables. (#958, 996, 997)
- Removed some redundant operations from the science code. (#1001)
- Removed redundant arguments for copydiag_3d (affects UM runs only). (#954)
- Added STASH code controls for a new scale-dependent gust diagnostic in the UM. (#984)
- Trivial change to USE statements relating to a UM change to atm_fields_mod. (#1003)

- Further work on CABLE front end, adding namelists *CABLE_PROGS* and *CABLE_SURFACE_TYPES*. (#940)
- Tidied to remove compiler warnings related to um_parvars (decomposition) and unused variables (NAG compiler), and to make JULES compliant with changes to UM debug compiler flags. (#963, 966, 1002)
- Added some missing platforms to the metadata for fcm-make (so those are available in rose edit). (#976)
- Fixed the make_jules_release script to push latest documentation onto the master on GitHub. (#957)
- The example namelists (difficult to maintain) and benchmarking suite (obselete) have been removed. (#928, 969)

1.11.3 Bugs fixed

- Bug fixes for the RothC-based soil biogeochemistry model (*soil_bgc_model* = 2) with *l_layeredc* = TRUE and *l_nitrogen* = TRUE. (#681)
- Prevented race condition in leaf_mod. (#1009)

1.11.4 Changes to testing

- Added NAG compiler to Met Office linux test suite. (#955)
- Fixed JULES rose stem build jobs on the VM. (#956)
- Updated NIWA rose-stem configuration for Maui HPC. (#965)
- Changed the path to data and libraries for rose stem testing on JASMIN. The default is the new jules group-workspace, and an environment variable JASMIN_JULES_BASE_DIR can be exported and picked up by rose stem. (#967)

1.11.5 Documentation updates

- Improved description of *l_embedded_snow*. (#916)
- Assorted clarifications in documentation and comments. (#980)
- Basic infrastructure provided to support future JULES Documentation Papers. (#968)
- Updates associated with many of the above changes, and release notes. (#995)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.12 JULES version 5.6 Release Notes

The JULES vn5.6 release consists of approximately 14 tickets from 11 authors, including work by many other people.

Full details of the tickets committed for JULES vn5.6 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #864.

1.12.1 Science changes

- Added the Farquhar model of photosynthesis for C_3 plants - see *photo_model*. (#864)
- Added the COARE algorithm for drag over sea, including functionality to reduce the drag at very high wind speeds - only affects runs with the UM. (#848)

1.12.2 General/Technical changes

- UM and JULES metadata consolidated for namelists *JULES_RADIATION* and *JULES_VEGETATION*. (#822)
- Preparatory work towards including irrigation demand in the UM. (#811)
- Allowed the dimension names *sclayer_dim_name*, *tracer_dim_name*, *bl_level_dim_name* to be read from namelist *JULES_INPUT_GRID*. (#937)
- Improved error handling in subroutines `set_levels_list` and `set_pseudo_list`. (#935)
- Updated the UM STASH diagnostics routines to use a modularised version of copydiag. (#938)
- Added further OpenMP optimisations for GA8 model routines. (#941)
- Additions of `#if defined(LFRIC)` to allow coupling of `surf_couple_extra` to LFRic. (#943)
- Altered OMP directives to remove data race conditions. (#952)
- Removed the requirement to have some environment variables that are used in build configs defined/initiliased at the app/suite level. (#939)

1.12.3 Changes to testing

- Updated rose stem testing to include JULES-ES-1.0 configuration. (#915)
- JULES can be built at the Bureau of Meteorology (Australia). (#930)

1.12.4 Documentation updates

- Updates associated with many of the above changes, and release notes. (#950)

Note that compilation of the User Guide now requires Sphinx 1.4 or higher.

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.13 JULES version 5.5 Release Notes

The JULES vn5.5 release consists of approximately 18 tickets from 14 authors, including work by many other people.

Full details of the tickets committed for JULES vn5.5 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #434.

1.13.1 Science changes

- Methane feedbacks from natural wetlands added to IMOGEN - see `land_feed_ch4`. This is done by adding an anomaly relative to the default emissions. Users should confirm that the wetland emissions are correct - see the notes under `land_feed_ch4`.
- Code changes for the GL9 configuration, including options to specify specific values for the roughness length of each Plant Functional Type (`l_spec_veg_z0`) and to impose a maximum-allowed value of the canopy heat capacity for vegetation (`l_limit_canhc`). (#903)
- Additional options for distributed form drag (*fd_stab_dep*) - not available in off-line JULES. (#870)

1.13.2 General/Technical changes

- Separate the calculation of plant-soil N fluxes from the updating of the soil stores - to allow the fluxes to be used with alternative soil models. (#651)
- Initial steps towards representing dry deposition in JULES: new namelists `JULES_DEPOSITION` and `JULES_DEPOSITION_SPECIES`. Note that deposition cannot yet be modelled in JULES. (#662)
- River routing code tidied. (#877)
- The interface (API) between the JULES program and the CONTROL subroutine now includes the atmospheric forcing variables as input and the gridbox mean surface flxues as output. This creates a clean "managed API" that can be used by parent models to call the JULES code at the CONTROL subroutine level. (#914)
- Added diagnostics for absorbed photosynthetically active radiation (*apar* and *apar_gb*) and gridbox mean leaf area index (*lai_gb*) - see *JULES Output variables*. (#614, 890)
- Corrected the units given for the ozone flux diagnostic (*flux_o3_stom*). (#905)
- Improved computational performance of ice-related and other routines. (#866, 894)
- Removed code only used for UM (atmospheric model) diagnostics; now using the sf_diag structure. (#899)
- Changes for UM diagnostics - to accommodate the fact that copydiag is now in a module. (#908)
- Moved the CABLE soil parameters to a separate namelist `CABLE_SOILPARM`. (#900)
- Improvements to code involved in creating the JULES release, including the documentation. (#893)

1.13.3 Bugs fixed

- Removed use of uninitialised memory in RFM river routing scheme. (#896)

1.13.4 Documentation updates

- Updates associated with many of the above changes, and release notes. (#924)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.14 JULES version 5.4 Release Notes

The JULES vn5.4 release consists of approximately 29 tickets from 13 authors, including work by many other people.

Full details of the tickets committed for JULES vn5.4 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #872.

1.14.1 Science changes

- Improvements to fire-related vegetation mortality, including the addition of a PFT-specific fire mortality parameter *fire_mort_io* (previously mortality was taken directly from the burnt area as diagnosed by INFERNO and did not vary by PFT). (#872)
- Stomatal conductance can be modelled following the approach of Medlyn et al. (2011), via the switch *stomata_model*. A single-parameter version of the model is coded, requiring the PFT-specific parameter *g1_stomata_io*. (#766)

1.14.2 General/Technical changes

- The RFM river routing scheme is now available to the UM (atmospheric model), and both standalone and UM runs use the same code. See *JULES_RIVERS*. (#876)
- The *JULES_RIVERS* namelist now controls river hydrology in both standalone and UM-coupled modes. (The UM namelist 'run_rivers' has been removed.) (#867)
- The surface conductance (*gs*) is now part of the specification of the initial state (and included in dumps) only when it is required (i.e. only if *can_rad_mod* = 1; see *List of initial condition variables*). (#859)
- Use `swap_bounds` routine(s) from `halo_exchange_mod` module (not the old 2C subroutine) in UM runs. (#367)
- Extensive refactoring of `surf_couple_extra`, including removal of `ifdef` in argument list. (#806, 833)
- Tidied/refactorised the photosynthesis code. (#817)
- Improved checking and reporting of the IMOGEN setup. (#850)
- Tidied the header of `control.F90`, removing duplicate and unused variables. (#873)
- Access subroutines `set_levels_list` and `set_pseudo_list` using modules, removing the need for `DEPENDS ON`. (#880)
- Improved performance of land surface routines in RA and GA configurations. (#861)
- Set up CABLE directory structure and initialise essential variables for *surf_couple_radiation*. (#799)
- Allowed variables used in the build process to have platform-specific defaults which can be overridden by the user. (#853)
- Met Office Cray users: Direct extract of code to the Cray is the default for meto-xc40-cce builds. Users are encouraged to remove fcm_make2 tasks and set `JULES_REMOTE = local` to take advantage of faster end-to-end compilations and reduce load on the HPC. Set `JULES_REMOTE = remote` to retain builds which require an fcm_make2 task. See *Environment variables used when building JULES using FCM make*. (#854)
- Reviewed and simplified fcm-make metadata compulsory variables and made current apps validate. (#855)
- Improved selected metadata in the *JULES_SOIL_BIOGEOCHEM* and *JULES_SOIL_ECOSSE* namelists to prevent errors when using the Rose GUI. (#862)

1.14.3 Bugs fixed

- Fixed the radiatively-coupled roof in MORUSES, using the temporary logical *l_fix_moruses_roof_rad_coupling*, in the new namelist *JULES_TEMP_FIXES*. The supposedly radiatively-coupled roof is in fact **uncoupled** without this bug fix. (#610)
- Corrected initialisation of frozen/unfrozen soil - no longer assumes constant soil properties with depth. (#749)
- Removed a bug in the snow scheme when *l_point_data* = TRUE and *can_model* = 4: the snow covered fraction formulation is now only used for tiles that do not use the snow canopy option (see *cansnowpft*), rather than for all tiles. (#879)
- Prevent out-of-bounds operations in sf_exch. (#846)
- Ensure that *ntype* is set before use in UM model runs. (#878)

- Corrected the units of the ocean near-surface chlorophyll content (used in the calculation of the ocean surface albedo), using the temporary logical *l_fix_osa_chloro*. Only affects runs with the UM. (#874)

1.14.4 Changes to testing

- Rose-stem fcm-make tasks will ignore lock files that would otherwise prevent retriggering. (#860)
- Expanded coverage of the rose-stem metadata validation test to include more apps. (#886)
- Upgraded `suite_report.py`. (#889)

1.14.5 Documentation updates

- Updates associated with many of the above changes, and release notes. (#881)
- Example namelists point_mead_2_crops have been updated to be consistent with the published JULES-crop runs in Williams et al. (2017).

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.15 JULES version 5.3 Release Notes

The JULES vn5.3 release consists of approximately 27 tickets from 17 authors, including work by many other people.

Full details of the tickets committed for JULES vn5.3 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #742.

1.15.1 Science changes

- Improved initialisation of the surface exchange iteration (`cor_mo_iter` = 4). (#742)
- Removed canopy radiation options (`can_rad_mod`) 2 and 3. (#791)
- Added nitrification, denitrification and leaching to the ECOSSE soil model (which is not yet ready for use). (#781)
- Allowed the use of a variable Charnock parameter, provided by a wave model via coupling, instead of a constant value, in runs of the UM (atmosphere-ocean model). See iseasurfalg = 4 or 5 in the jules_sea_seaice namelist. (#797)

1.15.2 General/Technical changes

- Coupling of river outflow from river grid to NEMO (ocean) grid via a 1D array. (#624)
- Added PBL gustiness parameter to namelist (`beta_cnv_bl`). (#742)
- Changes required to run the INFERNO fire model interactively in the UM. (#800)
- Initial modifications to allow irrigation code to be used in the UM. (#809)
- Optimised aspects of the canopy drag scheme. (#795)
- Resolved issues identified by OpenMP-related compiler warnings. (#712)
- Improved OpenMP coverage. (#723, 815)
- Improved performance in UM global ensemble configuration. (#820)
- Removed unused code from the surface scheme in standalone JULES. (#753)
- Removed idfefs from some of the surf_couple routine argument lists. (#830)

- Introduced a framework to move to shared metadata between UM and standalone for ease of converting to and from UM and standalone apps, to assist with configuration management. This introduces the new namelist `JULES_MODEL_ENVIRONMENT`. (#633)
- Ported to NIWA's XC50 platform. (#814)
- The meto-linux fcm-make configs have been converted to RHEL7. All Linux rose-stem tasks will run on RHEL7 SPICE nodes. Those using a meto-linux-intel-mpi build outside of rose-stem will need to set *ROSE_LAUNCHER_LIST = mpiexec.hydra* or make an equivalent change to their personal *rose.conf* file, and include the full path to the required MPI-enabled compiler. Refer to the *runtime-linux-intel.rc* file in rose-stem for details. (#835)
- Corrected bug in the make_jules_release script. (#796)
- Miscellaneous administrative changes. (#807, 839)

1.15.3 Bugs fixed

- Corrected a bug in the scalar roughness length diagnostic over sea when using anything other than a fixed roughness length, i.e. anything other than iseasurfalg=0. (#794)
- Fixed calls to mask compression routines in UM-only code. (#826)

1.15.4 Changes to testing

- A test of the canopy drag scheme, loobos_vegdrag, based on loobos_gl8 has been added to rose-stem. (#795)
- Updated *umdp3_fixer.py* to run on *.inc files, and included these in the rose stem test. (#776)
- Implemented code to align continuation ampersands in column 79 as part of *umdp3_fixer.py*. (#823)
- The Met Office rose stem suite now runs all Linux tasks on SPICE, and the metadata checker task is now included in the JASMIN rose stem suite. (#819)
- Correction for rose stem on MONSooN. (#852)

1.15.5 Documentation updates

- Updates associated with many of the above changes, and release notes. (#836)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.16 JULES version 5.2 Release Notes

The JULES vn5.2 release consists of approximately 52 tickets from 25 authors, including work by many other people.

Full details of the tickets committed for JULES vn5.2 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #754.

1.16.1 Science changes

- Introduced option for vegetation canopy drag with optional correction for the roughness sublayer - see `l_vegdrag_pft` and `l_rsl_scalar`. (#754)
- Soil decomposition added to the code for the ECOSSE soil model (which is not yet ready for use). (#570)
- Extension of the screen temperature decoupling diagnostics to screen humidity - only recommended for runs with the UM (atmospheric model). (#508)
- Added a new option to the sea albedo calculation to simulate the effect of freezing (sea-ice) below 271 K. (#770)

1.16.2 General/Technical changes

- Enabled support for new routines to calculate qsat (the saturated water mixing ratio) which are now the default for standalone JULES. (#685)
- Improved interface checking in the surface, fire, FLake and river routing routines. (#678, 728, 729)
- The clay ancillary variable can now have multiple layers (note these are the soil biogeochemistry layers, not soil moisture layers). Users should note that an existing run with `l_layeredc` = T that tries to read a single-layered clay variable from a file with `const_z` = F will no longer work; a multi-layered clay field must be provided in this case. All other configurations can be updated. (#687)
- Added further IMPLICIT NONE statements and fixed subroutine interface issues. (#737)
- Minor modifications to IMOGEN (#430).
- Removed the l_flux_bc switch from the UM-coupling argument list and standalone code. (#775)
- Improved use of coupled model diagnostic code in standalone runs. (#740)
- Diagnostics from INFERNO (interactive fire model) made available to the UM. (#552)
- JULES parameters included in the Random Parameter (RP) scheme (for UM runs). (#675)
- Rationalised some of the code for the reading and writing of dumps. (#763)
- Alignment of JULES and UM urban control and initialisation code. (#319)
- Reduced model overhead when running DrHook profiling. (#782)
- Fixed oddities found while investigating the use of CamFort. (#769)
- JULES source code fully compliant with the Fortran 2003 standard. (#711)
- Corrections to code comments and other minor changes. (#725, 690)
- Clarified units of variables in the vegetation code. (#741)
- Enable the reading of PFT and soil parameters for CABLE runs via the JULES_PFTPARM_CABLE and JULES_NVEGPARM_CABLE namelists. (#694, 748)
- Minor edits to ensure future compatibility of .inc files with the umdp3_fixer used in Rose stem tests. (#762)

1.16.3 Changes to testing

- Rose stem testing working on JASMIN. (#744)
- Improved the output of the umdp3 checker task in rose stem. (#764)
- Rose stem testing added for IMOGEN and GL7 and GL8 configurations. (#706, 648, 773)
- Added an OMP vs no-OMP Rose stem test for the ukv config to the MO XC40 and virtual machine platforms. (#732)
- Allowed the option of setting JULES_REMOTE and JULES_REMOTE_HOST when running on the Met Office Cray (meto-xc40-cce). (#755)
- Resolved oversubscription problems and rationalised the meto-linux rose stem. (#783)

1.16.4 Bugs fixed

- Fix to ensure TRIFFID competition does not try to access non-existent surface types. (#647)
- Fixed array/scalar mismatch in arguments to vegcarb. (#682)
- Corrected the dimensions given to the frac_prev array in *lotka_eq_jls.F90* (for runs with `l_trif_eq` = T and `l_ht_compete` = T). (#765)
- Fix to prevent floating point errors with CABLE. (#694)
- Use NINT to guard against imprecision in REAL/INTEGER conversion in routing code. (#726)
- Fixed bugs relating to windspeed-dependent unloading of snow from vegetation (UM only) and allowing soil rate modifier diagnostics in standalone runs. (#740)
- Correction related to indexing of snow fields in reconfiguration (UM only). (#676)
- Fixed certain snow diagnostics (UM stash fields 8,578 to 8,583). (#720)
- Correction to logic for canopy parameter updating (UM only). (#746)
- Example namelists updated for vn5.1. (#722)
- Fix of host specification in *runtime.rc* for site cehwl1. (#731)
- Updated the configuration for University of Exeter (*uoe-linux-gfortran.cfg*). (#735)
- Fix so rose stem IMOGEN tests work at NCI. (#792)

1.16.5 Documentation updates

- Removed the science configurations section from the documentation. (#736).
- Updated the documentation (mainly release notes and hydrological terminology). (#738, 745)
- Updated documentation of fcm-make JULES_PLATFORM environment variable. (#739)
- Updated hyperlinks to Rose and FCM in the documentation. (#786)

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.17 JULES version 5.1 Release Notes

The JULES vn5.1 release consists of approximately 40 tickets from 17 authors, including work by many other people.

Full details of the tickets committed for JULES vn5.1 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #533.

1.17.1 Science changes

- Addition of river overbank inundation module (for diagnostic calculation of overbank inundation fraction) - see l_riv_overbank in namelist *`JULES_RIVERS`* and namelist *`JULES_OVERBANK`*. (#679)
- Changes to the layered soil biogeochemistry model: (i) the mean soil temperature over the TRIFFID time step is now used to determine whether a layer is unfrozen (ii) mixing is considered when calculating the final respiration. (#663)
- Options for improved treatment of thin snow comprising the introduction of basal melting of thin snow layers on warm ground. See `i_basal_melting_opt`. (#533)
- Added capability to include latest harvest date to crop ancillary. See *List of spatially-varying crop properties*. (#653)
- Account for crop harvest and fire in the carbon conservation diagnostics. (#476)
- Carbon conservation diagnostic now only calculated on TRIFFID timesteps. (#643)

- Added the JULES-standalone version of improved saturation vapour pressure (qsat) calculations. (#635)
- Introduced basic slab ocean by allowing the model to update the sea surface temperature based on the energy balance, in the same way as it does for land and sea-ice. Also allows for a fixed sea surface albedo (see *fixed_sea_albedo*), rather than the parameterised options. (#642)

1.17.2 General/Technical changes

- Wetland CH_4 emission parameters have been added to *JULES_SOIL_BIOGEOCHEM*. Parameters are *t0_ch4*, *const_ch4_cs*, *const_ch4_npp*, *const_ch4_resps*, *q10_ch4_cs*, *q10_ch4_npp*, and *q10_ch4_resps*. (#483)
- i/o work to allow soil tiling to function correctly. (#684)
- Allow output of rate modifier diagnostics needed for offline spin up of soil carbon pools. (#589)
- Improvements to intermittent sampling for output. (#605)
- Removed repeated calculation of vegetation stocks of C and N from TRIFFID. (#618)
- Modularised JULES vegetation subroutines to enforce stricter interface checking. (#668)
- Replaced integer constants used with *ch4_substrate* (the choice of substrate for wetland methane) with parameters. (#628)
- More time constants replaced by parameters. (#625)
- Namelist *lsm_id* added to allow the selection of the land surface model: JULES or CABLE. Note that the CABLE science routines have not been implemented yet. (#656)
- Redundant “include” files removed and rationalisation of other modules. (#689)
- Made the JULES source code (as close as possible to) compliant with the Fortran 2003 standard. (#699)
- Added OpenMP to the snow scheme to improve parallel performance. (#638)
- IMOGEN identified as a new science module, and other changes to list of module leaders. (#686, 666)
- Added basic MORUSES rose stem test (upgraded UKV science to PS39) and enhanced metadata. (#289)
- Added a new script to rose stem to ensure apps are consistent with rose metadata. (#645)
- Expanded the fcm-make metadata to make it easier to apply additional compiler flags to builds. (#632)
- Implemented a timeout for the JULES rose-stem in instances where an app/task “submit-fail” or stalls. (#672)
- Updated settings for JULES on Met Office XC40 (standalone suites should be updated where necessary to pick up the new settings) and Met Office linux testing to use ifort 16.0. (#630, 697).

1.17.3 Bugs fixed

- Fixed bug in soil hydrology to prevent water erroneously coming out of the bottom of the soil. Use *l_holdwater* = T to allow water to be held on an impermeable layer. (#171)
- Bug fix affecting litterfall N flux when *l_landuse* = T. (#617)
- Bug-fix in the calculation of the albedo when *l_embedded_snow* = T. (#533)
- Corrected a scalar/array mismatch in the arguments to the *plant_growth_n* subroutine. (#670)
- Fix to allow initialisation of CO_2 from a dump file. (#680)
- Minor fix for vegetation competition in runs with *l_ht_compete* = F and *l_trif_crop* = F. (#627)
- Corrected units for parameters *dfp_dcuo* and *eta_sl* in comments and documentation. (#646, 659)
- Minor correction to metadata for *JULES_SOIL_PROPS* namelist. (#669)
- Fixed an allocation bug in the UM (#665).
- Bug in tiling that could affect lake quantities (in UM runs using FLake). (#641)
- Fix of a minor OpenMP race condition. (#674)

- Moved an ALLOCATE statement to fix a memory bug in the UM affecting ESM runs. (#639)

1.17.4 Documentation updates

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.18 JULES version 5.0 Release Notes

The JULES vn5.0 release consists of approximately 34 tickets from 12 authors, including work by many other people.

Full details of the tickets committed for JULES vn5.0 can be found on the JULES shared repository Trac system.

This release was unusual in that it was closed to science tickets so as to allow work of a more technical nature, principally ensuring compliance with the coding standards and improved optimisation.

The extensive styling changes that were introduced at vn5.0 have made minor changes to a large number of lines of code, meaning that updating your work from an older branch may not be as straightforward as with most release cycles. For advice on how best to upgrade your branch to vn5.0 see Notes on moving to vn5.0.

Ticket numbers are indicated below, e.g. #230.

1.18.1 General/Technical changes

- Applied a code style-checking script, also included as part of rose stem tests. (#230, 387, 593)
- Modularise the root_frac() subroutine to improve interface checking. (#571)
- More use of OpenMP threading to surface and soil code to improve performance. (#575, 600, 607)
- Various code changes to prevent or warn if an inconsistent combination of vegetation flags and parameters is used. When TRIFFID is on and the phenology model is off (not recommended), LAI is now set to the balanced LAI rather than taken from the (*JULES_PFTPARM*) namelist. (#592)
- Remove references to outdated or deprecated functions (#577, 585), and other technical improvements. (#596, 616)
- More variables given initial values. (#579)
- Reduced size of outputs from rose stem tests. (#580)
- Superfluous messages from vegetation code removed. (#581)
- General code improvements to meet coding standards and use a consistent style. (#574, 576, 578, 584, 594, 620, 621)
- Namelists are printed before checking for errors, for easier debugging. (UM only, #582)
- Snowdepth diagnostic made available in the UM. (#569)
- Module leadership clarified across the code base. (#611)
- Unnecessary subversion properties removed. (#597)

1.18.2 Bugs fixed

- Fixed minor bug in variable names when soil moisture prescribed. (#573)
- Prevent duplicate names for output streams. (#590)
- Fixed bug in vegetation competition routine lotka_eq. (#603)
- Fixed bug in output variable smc_avail_top. (#604)
- Fixed minor bugs in use of l_trait_phys. (#613)
- Corrected wood product pool diagnostics in the UM. (#587)
- Bugs fixed in soil respiration, and in carbon conservation diagnostics, for runs with N limitation. (#595)
- Enables ability to perform bit comparable NRUN in the UM. (#612)
- Fix JULES rose stem so that it works on the MO XCS-C system (aka MONSooN). (#615)
- Fixed broken hyperlinks in the user guide. (#636)

1.18.3 Documentation updates

Documentation can be viewed on the github page http://jules-lsm.github.io/.

1.19 JULES version 4.9 Release Notes

The JULES vn4.9 release consists of approximately 60 tickets from 19 authors, including work by many other people.

Full details of the tickets committed for JULES vn4.9 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #262.

1.19.1 Science changes

- New parameter `s_pdm` represents the minimum soil wetness below which there is no saturation excess surface runoff from the PDM model. s_pdm can be made slope-dependent using `l_spdmvar`, with additional parameter `slope_pdm_max`. (#262)
- Extensions to the parameterisation of moisture stress on vegetation. There is now (a) the option of a stress function that is piece-wise linear in soil potential (`fsmc_shape` = 1) rather than soil volumetric moisture content (`fsmc_shape` = 0) and (b) the option of specifying pft-dependent soil potentials (`l_use_pft_psi`) at which the plant starts to experience water stress (`psi_open_io`) and where the plant is fully stressed (`psi_close_io`). (#541)
- Methane emissions can now be calculated from layered soil temperature (see `l_ch4_tlayered`). Methane flux can be removed from soil carbon stocks to close the carbon budget (see `l_ch4_interactive`). The choice of substrate used for methane production is now controlled using `ch4_substrate` instead of l_wetland_ch4_npp. (#468)
- Updates to the INFERNO fire model (see `l_inferno`) including burning of the carbon in litter/soil pools. New diagnostics added including flammability (see *JULES Output variables*). (#502)
- Extension to the scheme for calculating subgrid-scale snowpack properties and surface mass balance fields for icesheet coupling (see `l_elev_land_ice`). Non-glaciated tiles (`elev_rock`) can now co-exist with the ice tiles (`elev_ice`) in the elevation classes, allowing for fractional ice extent in a gridbox. (#294)
- Allow soil moisture to be prescribed on a subset of soil levels (see `prescribed_levels`). (#487)
- Improvements to the layered soil C and N model (see `l_layeredc`) which is now fully ready for use. (#454, #526)
- Layered C allowed with single-pool soil model. (#526)

- Preparatory work for the ECOSSE soil biogeochemical model. (#444, #518)
- New functionality to limit the drag over the sea at high wind speeds (only for coupled models). (#543)

1.19.2 General/Technical changes

- Further preparatory work for soil tiling, including i/o.
- Allocated arrays are given initial values and improved error messages from memory allocation.
- Further C fluxes and diagnostics made available to the UM.
- *fsmc_p0_io* added to UM namelists.
- Various UM-related improvements, including: single switch to allow runs with mixing ratio; avoiding hard-wired logical argument to swap_bounds; removed old logical related to "New dynamics"; removed the stash super array; removed unused array bounds.
- Wrapped non-LFRIC-compliant code using a preprocessor directive.
- General code developments and improvements to meet the coding standards.
- Added fcm make and rose stem configurations for building and testing JULES at NCI.
- Added Rose stem tests for TRIP rivers running with 2D data and RFM rivers. (#539, #567, #568)
- Add more information to trac.log by improving suite_report.py.
- rose-stem supported for MONSooN xcs-c.
- Various changes for Met Office, including nightly testing on xcs.
- New tutorial group for rose-stem.
- Change to suite-report.py for use with cylc 7.
- Updates to the app update script.

1.19.3 Bugs fixed

- Corrected mismatches between certain subroutine interfaces and subroutine calls. (#471, #544)
- Fixed bug in reading of certain soil tile variables. (#498)
- Fixed bugs in TRIFFID litter fluxes. (#504)
- TRIFFID altered to do fewer calculations on points without veg or soil. (#509)
- Tidied up code issues between UM and JULES. (#522)
- Added interactive nitrogen to the loobos_jules_es_1p6 test and initialised more TRIFFID variables. (#512, #520)
- Bug fix for regridding between land and river routing grids for regular lat/lon river grids where the land model grid is defined as 1d land point only grids. (#524)
- Fixed uninitialised variables in surf_couple_extra_mod.F90. (#546)
- Bug fix in init_rivers_props.inc. (#539)
- Bug fix for crop ancillary variables in read_dump and write_dump. (#551)
- Corrected a bug in the sea ice albedo scheme, which affects the calculation of bare ice albedo when multilayer thermodynamics are used in coupled UM-JULES-NEMO-CICE. (#547)
- Bug fix to variable intent in surf_couple_implicit. (#542)
- Bug fix for UM runs relating to clash between STASH items (#557)

1.19.4 Documentation updates

Coding standards, and documentation can be viewed on the github page http://jules-lsm.github.io/.

1.20 JULES version 4.8 Release Notes

The JULES vn4.8 release consists of approximately 77 tickets from 26 authors, including work by many other people.

Full details of the tickets committed for JULES vn4.8 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #400.

1.20.1 Science changes

- Changes to calculate transpiration on tiles and as a gridbox mean. A resistance factor based on stomatal resistance excluding soil is calculated and then multiplied by evapotranspiration. This affects the diagnostics ‘et_stom’ and ‘et_stom_gb’. (#400)
- Addition of interactive fire/vegetation, selected using *l_trif_fire*. Fire disturbance modifies vegetation dynamics and can be modelled by the INFERNO fire model or prescribed via an ancillary file. This is a preliminary version that will be developed further. (#456)
- Added option to allow downward longwave and net shortwave radiation to be used to force the model (see *List of JULES forcing variables*). JULES now stops if too many or incorrect radiation variablse are provided (rather than carrying on if a valid combination was found). (#409)
- Added an option for the treatment of graupel in input to snow scheme (primarily for the UM; see *graupel_options*). (#414)
- Looping required for soil tiling functionality added in a disabled state (nsoilt still hard-coded to 1). (#379)
- Added the ability for hydrol_jls.F90 to work with tiled runoff once soil tiling is enabled in a later change. (#341)

1.20.2 General/Technical changes

- Refactoring of albpft_jls.F90 to be more efficient.
- Improved code to calculate litter, landuse change and harvest fluxes.
- New namelist *JULES_SOIL_BIOGEOCHEM*. The soil and vegetation components of TRIFFID have been separated.
- *total_snow* defaults to .FALSE.
- *dump_period* controls the frequency with which dumps are written.
- River routining changes: user initialised surface and sub-surface storage and flows for RFM river routing. See *List of initial condition variables*. Two options for standalone river routing: ‘rfm’ or ‘trip’ in namelist *JULES_RIVERS*. New remapping utilities provide efficient translation between land points and river points vectors.
- New and added diagnostics for UKESM, CMIP6 ‘NDVI_land’. See *JULES Output variables*.
- Improved error checking and error messages.
- Print statement improvements.
- Urban namelists prefixed with ‘jules’.
- Tidy up unused variables and an unused dummy argument in the FLake code.
- General code developments and improvements to meet the coding standards.
- A minor change to communication routines’ API.

- OpenMP developments.
- Rationalisation of UM ancil routines, part 1 (#346)
- Removed redundant code from the soil ancillary reading code. (#450)
- Rose stem configurations for JASMIN, NIWA, CEH, Exeter Uni, remote JASMIN and MetO XCS computer. Ensure that the same tests are run at all sites. cylc7 compatable
- New or improved rose stem tests.
- Improvements to create-rose-app and suite_report.py

1.20.3 Bugs fixed

- Bug in the argument list to irrig_dmd, where the surface-tiled irrigated fraction was passed instead of the gridbox mean. (#389)
- Various fixes related to argument INTENTs (snow #396; soil respiration #392, h_blend_orog #452, surface flux code #412).
- Bug fixes for INFERNO fire model (soil carbon #415, incorrect rainfall #390, minor fixes #371).
- Bug fix in TRIFFID so soil nitrogen with l_trif_crop does not depend on processor configuration (#372)
- Bug fix for soil inorganic N in UM runs (#395)
- Bug fix in hydrology for when `l_wetland_unfrozen` = .TRUE. (#473)
- Initialisation of TRIFFID diagnostics (#313, #474, #479)
- Minor update to correct standalone river routing grid definition for non-regular grids (e.g. UKV variable resolution). (#410)
- Bug fix in the calculation of NPP in TRIFFID with N limitation. (#308)
- Fixed error in frequency of calls to phenology in long standalone runs. (#421)
- Bugs fixed so IMOGEN will restart more accurately from a dump. (#420)
- Bug fixes and improvements for layered soil CN model (#394, #407)
- Bug fix for albpft_jls.F90 (#458)
- Bug fix to allow compilation without netCDF (#464)
- Bug fixes for metadata and upgrade macro (#404, #459, #490)

1.20.4 Documentation updates

Coding standards, and documentation can be viewed on the github page http://jules-lsm.github.io/.

1.21 JULES version 4.7 Release Notes

The JULES vn4.7 release consists of 47 tickets from 19 authors.

Full details of the tickets committed for JULES vn4.7 can be found on the JULES shared repository Trac system.

Ticket numbers are indicated below, e.g. #265.

1.21.1 Science changes

- Enable soil tiling by the extraction of key calculations. These include: Infiltration rate and Soil moisture availability factor (beta). (#265)
- Modifications to the rate of growth of snow grains - it uses the ET scheme of Taillandier (2007), JGR, 112, F03003 for the rate of growth and to relayer the grain size using its inverse as this is more consistent with conservation of SSA reported by Gallet et al. (2011). (#298)
- JULES-CN: Soil_CN ratio changed from hard-wired to the namelist in prep for a PPE of JULES and JULES-CN the bio and hum N pools change from being prognostic to diagnostic via the soil bgc at the start of this routine and during initialisation. (#309, 288)
- Add new irr_crop option: irr_crop = 0: continuous irrigation (i.e. the effectively the crop season is defined to last all year). It does not depend on crop characteristics (unlike irr_crop=2, which uses the crop model, or irr_crop=1, which estimates a typical crop season for that gridbox). (#312)
- Diagnostics for individual components of snowpack mass balance, as specified by ISMIP6. (#314)
- Diagnostics for components of surface radiation on land tiles. Requested by ISMIP6 for driving standalone icesheet models. (#315)
- User initialised river storage. When dump_file=T, and use_file=T for rivers_sto_rp, then rivers_sto_rp needs to be in the dumpfile. When use_file=F for rivers_sto_rp, then rivers_sto_rp can be set to a constant value in this namelist. When dump_file=F, the rivers_sto_rp is initialised to zero. (extra log message to say that rivers_sto_rp is initialised to zero in this case). Therefore a dump file from a run without l_rivers=T and rivers_type='trip' can now be used to initialise a run with l_rivers=T and rivers_type='trip'. (#316 and #329 doc)
- For fsmc_mod = 1, change the water extraction pattern so that it is proportional to plant available water rather than total water in the soil layers. Note: fsmc_mod=1 is not the recommended value in the JULES manual and it is not currently use in any of the documented configurations. This part of the code is also going through a detailed review as part of the soil moisture stress on vegetation group, which includes documenting the current status. (#320)
- Add a new cpft-dependent input variable initial_c_dvi_io to specify when the crop should be initialised. (#324, 356 doc)
- Allow perturbations to driving data, by specify an amount to be added to driving temperatures and/or an amount to multiply the driving precip variables by. This is achieved by adding a switch (l_perturb_driving) and two input variables (temperature_abs_perturbation and precip_rel_perturbation) to the JULES_DRIVE namelist. When l_perturb_driving is set to true, an amount (positive or negative) can be added to the driving temperature (temperature_abs_perturbation) and the precipitation variables can be multiplied by a factor (precip_rel_perturbation). (#326)
- Fluxes JULES rose stem tests added: Some new GSWP2 tests for forecast configurations, New diagnostics to many existing tests, Switches on profiles that were written but disabled in carbon cycle tests, Tweaks to how we run on SPICE. (#330)
- Change the defaults for the npft=9, ncpft=4 case in the rose upgrade macro vn44_t136 so that first 5 pfts are the same as the npft=5 case and the C3/C4 crop tiles have the same values as the C3/C4 grasses. (#336)
- Allow sthuf (soil moisture) to be prescribed. (#348)
- Soil refactor, following on from adding module statements etc to radiation and snow, this works on src/science/soil to add MOUDLE statements and INTENTS. (#351)
- Update to metadata for jules_rivers_props so when grid_is_1d is false the following values in jules_rivers_props; nx_grid, ny_grid, reg_lat1, reg_lon1, reg_dlat, reg_dlon are required when running in parallel. Added a log message warning that the values are not used if grid_is_1d is false and the run is in parallel. (#293)
- Add the soil tile dimension to JULES as a hard-coded singleton. (#305)
- Fixed a bug in TRIFFID which causes loss of bit comparison that occurred when L_TRIF_CROP and L_NITROGEN were both TRUE: the soil nitrogen prognostics (stash 442, 443, 446) became dependent on the PE configuration. (#372)

- Modularise and header refactor science/radiation Adds module and intents to the radiation code Removed implicit RESHAPES of some variables through subroutine calls and therefore removes nearly all the complicated dimensionalities in this area (ij, pfield, land_pts), making the code simpler. (#253)
- Add vertically discretised soil C and N to TRIFFID. Adds a dimension to existing prognostics and discretizing existing code. The scheme is extended to link rooting profiles to availability and uptake of N requiring additional prognostics. (#288)

1.21.2 General/Technical changes

- Improve coding standard docs.
- Move from includes to use modules for ccarbon.h Retire l_endgame as we only use endgame in the UM.
- Fix warnings in the fcm make log.
- OpenMP improvements.
- Add valgrind profiling as execution option in the Rose GUI.
- Extra documentation on; namelist order, Update to release notes.
- Nightly rose stem test added to the MO system to test the head of the trunk nightly.
- Replace all ENDIF, ENDDO, IF(, MAX(, MIN(, EXP(and GAMMA/gamma to r_gamma with the working practices syntax.
- Use “rose config dump” on the whole repository to tidy up the rose (Python) files.
- Update cylc5 syntax to cylc6 (Python).
- Improve the create_rose app script. It now takes 5 arguments, vn_from, vn_to, namelist_path, suite_name and jules_dir.

1.21.3 Bugs fixed

- Fixed the variable in the metstats_timesteps subroutine that was being incorrectly set for first and last second of day, which lead to gaps in the fire indices along some lines of longitude. (#323)
- Fixed errors in UM GA7 AMIP code to run rose stem app with “rigorous” optimisation settings on SPICE. In particular: Formatted internal writes with incorrect format statements; ALLOCATEd variables not being DEALLOCATEd. (#328)
- Fixed example put namelists. Updated parameters can_struct_a_io, fsmc_mod_io, and fsmc_p0_io in all example directories, and added missing parameters in the loobos_point_9pfts directory. (#354)
- Clay frac bug, IF test for l_triffid=.false. added around clay_gb so that the test does not fail as clay_gb has not been populated as triffid is not run and therefore not clay_gb is not initialised and populated. (#307)
- taux (wind stress) output inconsistency fixed by the initialisation of two variables in surf_couple_extra: cq_cm_u_1(:,:) = 0.0 and cq_cm_v_1(:,:) = 0.0 (#339)
- Set z0_surft and lit_c_pft to zero in allocate_jules_arrays as they were not allocated and caused KGO failures. (#365)
- Fixed calc_fsat so that values are not divded by small numbers. (#342)
- Fixed the bug in multi-layer snow use of tile_map. It now checks to see if it is valid for the input dump configuration and converts from mapping tile IDs to pseudo levels. (#302)

1.21.4 Documentation updates

Coding standards, and documentation can be viewed on the 'github page <http://jules-lsm.github.io/>_'.

1.22 JULES version 4.6 Release Notes

The JULES vn4.6 release consists of 43 tickets from 22 authors across 52 commits.

Full details of the tickets committed for JULES vn4.6 can be found on the JULES shared repository Trac system.

1.22.1 Science changes

- Multiple ice tiles in a gridbox to simulate snowpacks at different elevations - see *l_elev_land_ice*
- Modifications to snowpack physics to better represent deep, compact firn/snow on ice sheets - description at *l_elev_land_ice*
- Option to link whole-plant maintenance respiration to soil moisture stress - see *l_scale_resp_pm*
- Read clay content of soil for soil carbon decomposition model from ancillary file - see *JULES_SOIL_PROPS*
- Improved climate downscaling physics for ice elevation tiles
- Calculate FAO Penman-Monteith evapotranspiration for reference crop - see diagnostic fao_et0
- Diagnostic form drag for sea ice (UM only)
- Calculation of new sea ice variables required for CMIP6
- Implement a canopy clumping factor - see *can_struct_a_io*
- Allow for non-istropic scattering in plant canopies - see *l_niso_direct*
- Increased flexibility to represent soil moisture stress on vegetation - see *fsmc_mod_io* and *fsmc_p0_io*
- Improved parameterisation of crop leaf senescence
- New crop harvest diagnostics
- Lake model FLake beneath multi-layer snow (UM only)

1.22.2 Technical changes

- JULES-C-1p1 Regression Tests
- Remove the UM_FLAKE CPP macro
- Move sorp and n_inorg_turnover to namelist to enable user input.
- VM rose stem bug fixed.
- Add support for rose-stem on MONSooN
- Move some of the hard-wired crop parameters to *JULES_CROPPARM*
- Remove UM descent.h include file and put values into JULES module descent.F90
- Fix race conditions and improve OpenMP DEFAULTs
- Modularise and header refactor science/snow

1.22.3 Bugs fixed

- Fixed GC3 tstar_sice bug
- Corrected canopy nitrogen profiles - see *l_leaf_n_resp_fix*. This increases plant maintenance respiration.
- Reduced stem respiration with trait-based physiology
- Reinstated missing veg parms, for trait initialisation.
- Soil respiration bug resolved
- Corrected mistake in merging of crop PFT changes
- Fixed N diagnostic to avoid runtime crashes
- Fixed calculation of dust deposition exchange coefficient
- Fixed wetland emission of methane with TRIFFID on
- Fixed aerodynamic resistance diagnostic (UM diagnostic)

1.22.4 Documentation updates

Coding standards, and documentation can be viewed on the 'github page <http://jules-lsm.github.io/>_'.

1.23 JULES version 4.5 Release Notes

The JULES vn4.5 release consists of 31 tickets from 19 authors across 35 commits.

Full details of the tickets committed for JULES vn4.5 can be found on the JULES shared repository Trac system.

1.23.1 Science changes

- UKCA dry deposition working with 13 surface tiles
- Check litter flux carbon balance
- Allow litter carbon fluxes from variable numbers of PFTs
- Improved seasonal cycle of soil respiration: switch l_soil_resp_lev2 alters how soil temperature and moisture are used for respiration calculation.
- Added parameters to trait physiology for nitrogen in wood and roots - see *nr_io*, *nsw_io* and *hw_sw_io*.
- INFERNO model of fire emissions and burnt area - see *l_inferno*
- Option to represent crops using triffid - see *l_trif_crop*
- Remove MORUSES hard-wired roof coupling
- Add diagnostic for canopy FAPAR (Fraction of Absorbed Photosynthetically Active Radiation).
- JULES-CN: enabled nitrogen limitation of NPP - see *l_nitrogen*
- Added the FLake lake model into JULES (for UM use)

1.23.2 Technical changes

- Variable renaming to support soil tiling
- Add JASMIN as a supported system for rose-stem jobs
- Fix Cray compiler warnings
- Protect print statements from the TRIP river routing code with PrintStatus

1.23.3 Bugs fixed

- Fix ozone diagnostics in JULES

1.23.4 Documentation updates

Coding standards, and documentation can be viewed on the 'github page <http://jules-lsm.github.io/>_'.

- Update to coding standards to reflect and protect variable name changes
- Represent crops using TRIFFID
- JULES-CN
- Nitrogen trait physiology
- Check litter C flux carbon balance
- kgC and kgN in netCDF units metadata
- adding nfita to hydrology namelist
- add FAPAR diagnostic

1.24 JULES version 4.4 Release Notes

The JULES vn4.4 release consists of 31 tickets from 17 authors across 35 commits. Two further tickets were removed from the release due to issues.

Full details of the tickets committed for JULES vn4.4 can be found on the JULES shared repository Trac system.

1.24.1 Science changes

- Add an option to set tile elevations to have absolute values above sea-level - see `l_elev_absolute_height`
- Adjustment to downward longwave radiation for elevated tiles - see `l_elev_lw_down`
- Nitrogen cycling - improved process representation for UKESM
- Use bare soil momentum roughness length, if supplied as an ancillary field
- Irrigation water taken first from deep soil layer then the river (code from the JULES Impact Model)
- Improvements to rivers_type='trip','rfm': storage in dump and partial parallelisation
- Alternative forms for methane emissions from wetlands, using different substrates
- Allow landuse with variable number of PFTs
- BVOC emissions allowed with trait physiology
- Change multi-layer snow indices of first layer

1.24.2 Technical changes

- JULES fcm-make configs updated to be incompatible with FCM 2015.07.0 and later
- Rose-stem support for alternative Rose, Cylc and FCM versions
- Addition of OpenMP directives to some JULES code.
- Code coverage metrics for rose-stem suite
- Preparation for JULES memory duplication
- Removed snow_grnd_gb from closures benchmark test

1.24.3 Bugs fixed

- Remove factor of 0.5 from snow albedo temperature dependence
- Fix a bug in the Nitrogen scheme for the implicit update of soil respiration when the soil C pool is exhausted.
- Fix drift in vegetation fractions
- Scaling bug in Taux
- Bug fix for irrig_water diagnostic - between crop seasons this is now zero
- Met Office VM: fix for update-jules-scripts file for JULES 4.3
- JULES namelist broadcasts are out of sync

1.25 JULES version 4.3 Release Notes

The JULES vn4.3 release consists of 36 tickets from 18 authors across 39 commits.

Full details of the tickets committed for JULES vn4.3 can be found on the JULES shared repository Trac system.

1.25.1 Science changes

- Enhancements to the multi-layer snow scheme for GL7.0 (Global Land configuration, version 7.0)
  - Addition of ET metamorphism
  - Infiltration of rain water into the snow pack
  - Albedo of snow and relationship to plant canopies
- Generalisation of the crop scheme to work with trait-based plant physiology and BVOC emissions
- Update to wetland scheme (see *l_wetland_unfrozen*)
- River routing updates to allow RFM with standalone JULES to be run with non-regular lat-lon grids
- New JULES-C configuration, the prototype configuration for UKESM1
- Sea-ice changes for GC3.0 (Global Coupled configuration, version 3.0)

1.25.2 Technical changes

- Revamp of compilation procedure (see *Building and running JULES*)
  - Changes to the environment variables used to specify a build
  - Option to extract and mirror on local machine, preprocess and build on a remote machine (e.g. Met Office Cray XC40)
  - Addition of "platform configurations", to reduce the number of environment variable definitions required to build on a known platform
- Ancillary data (e.g. fractional coverage, soil data) is now saved to the dump file
  - Each namelist in *ancillaries.nml* gets a new flag, `read_from_dump`, e.g. `JULES_SOIL_PROPS::read_from_dump`
  - A dump file can now be used initialise an entire run, including ancillaries (except for river routing, for technical reasons)
- Many additional `rose-stem` tests
- Replace testing for ice using `sm_sat` with logical arrays for soil and ice points
- Restructuring of `rose-stem` tests to allow for site configurations with more divergence between sites
  - As a result, JULES is now routinely tested on 3 platforms - Intel and gfortran compilers on Linux and CCE on the Cray
- Remove the hijacking of the ice tile as a second urban tile when using two-tile urban schemes in the UM
- Replacement of old include files with modules

1.25.3 Bugs fixed

- Fixed a long-standing off-by-one error in the instantaneous interpolation code (mode `i` - see *Temporal interpolation*)
- Several small fixes to soil carbon and vegetation code
- Fix in river routing for bit-comparison with different processor decompositions in the UM
- Fix for IMOGEN in parallel mode
- Fix initialisation of some diagnostics

1.26 JULES version 4.2 Release Notes

JULES version 4.2 is the first release where all development has taken place in the Met Office collaboration repository. On the whole, this has been a success, with module leaders beginning to use the Trac system to track and approve developments in their areas.

The JULES vn4.2 release consists of 35 tickets from 13 authors across 39 commits.

1.26.1 Science changes

- TRIP and RFM river routing (see *jules_rivers.nml*)
- Incorporation of widely used fire risk indices (see *fire.nml*)
- New soil thermal conductivity model that is more appropriate for organic soils (see `soilhc_method`)
- Addition of "bedrock" column beneath the soil column, in which only thermal diffusion occurs (see `l_bedrock`)
- New canopy radiation scheme in which the nitrogen follows an exponential decay (see `can_rad_mod`)

- Updates to trait PFTs (see *l_trait_phys*) to ensure that the dynamic and equilibrium solutions for vegetation fraction are equivalent
- Crop model now conserves carbon

1.26.2 Technical changes

- Added flag to force the model grid to be 1D (see *force_1d_grid*)
- Several new diagnostics added (see *JULES Output variables*)
- New rose-stem regression tests added
- All rose-stem tests migrated to use NetCDF and `nccmp`
- Retirement of several include files
- Removal of logging namelist - output location should now be controlled using pipes (i.e. `1>/my/file 2>&1`) or features of your `mpiexec` or `mpirun` program
- Additional consolidation of shared (i.e. standalone and UM) control and initialisation routines
- Optimisation of `sf_stom` and `leaf_limits`, resulting in ~20% speedup for *can_rad_mod* = 4

1.26.3 Bugs fixed

- Bug in calculation of `n_leaf` and `n_stem` in `sf_stom`
- Memory overwriting bug in TRIFFID
- Differing error message lengths in UM
- `hcons` now passed to the snow scheme instead of `hcon(:,0)`
- Several bug fixes for IMOGEN
- Patch to enable collective access for parallel NetCDF in NetCDF 4.2 onwards
- `soil_hyd` now declares `ksz` correctly (i.e. `ksz(npnts,0:nshyd)` instead of `ksz(npnts,nshyd)`) - this only affects runs where the soil properties vary for each layer
- Correction to the accumulation of `g_leaf_phen` in-between calls to TRIFFID

Full details of the tickets committed for JULES vn4.2 can be found on the collaboration repository Trac system.

1.27 JULES version 4.1 Release Notes

1.27.1 Irrigation demand

When enabled, irrigation demand adds water to the soil moisture up to the critical point, meaning that vegetation does not experience water stress. At the moment the amount of water added is not limited, although it will be limited in a future version.

There are two schemes that can be used to determine when to irrigate:

1. This method calculates optimum planting dates for non-rice crops using averages from driving data and so requires at least one year of driving data. It was written by Nic Gedney and is based on the crop calendar from Doell & Siebert (2002).
2. Uses development index (dvi) across all tiles from JULES-crop (written by Rutger Dankers). Maximum dvi must exceed -1 (indicates sowing) for irrigation to occur.

See the documentation for *l_irrig_dmd* for more details.

1.27.2 Carbon cycle developments

A number of carbon cycle developments are included in this release:

1. Changes to the competition code to allow for flexible, height-based competition in TRIFFID. See `l_ht_compete`.
2. Trait-based plant physiology that allows the plant physiology to be defined by parameters that are more readily definable from observations. See `l_trait_phys`.
3. Code for simulating land-use change (e.g. forest clearance) including product pools (previously implemented in HadGEM2-ES). See `l_landuse`.
4. Time-varying agricultural fraction now officially supported. Time-varying CO2 concentration is also supported with some caveats. See *the list of supported prescribed data variables*.
5. Switch to enable/disable the adjustment of fractions during initialisation. See `l_recon`.

1.27.3 Maximum and minimum output types

It is now possible to output the maximum and minimum value over the output period of any variable, e.g. monthly maximum.

See `output_type` for more information.

1.27.4 Changes to the coupling routines

The routines that couple the JULES science to the UM and to the standalone wrapper have changed - see `src/control/standalone/control.F90`.

This is mainly in order to simplify the coupling with the UM and to facilitate upcoming developments.

1.27.5 Bugs and other changes

- It is now possible to run with a fixed 365-day calender (i.e. no leap years) using the `l_leap` flag.
- Corrected temperature limitation on soil respiration to be consistent with HadGEM2-ES.
- Minor tweaks to the crop model.
- Improvements to the BVOC emissions model, including linking with UKCA via the UM for coupled studies.
- Changes to introduce the COARE algorithm for surface exchange over the ocean.
- Improved error messages for I/O errors (e.g. namelist reading)
- Fixed a bug where the start and end time of data are not initialised correctly when using the daily disaggregator.
- Fixed a bug in tilepts caused by the use of short-circuiting logic that is not supported by some compilers.

1.28 JULES version 4.0 Release Notes

1.28.1 JULES-Crop crop model

JULES vn4.0 sees the introduction of the JULES-Crop crop model. This has been the result of many years of hard work from Tom Osbourne et. al. at the University of Reading.

A lot of the work done in getting it ready for the trunk and testing was done in the Met Office by Karina Williams and Jemma Gornall.

1.28.2 Daily disaggregator for forcing data

JULES can now be driven with daily forcing data, and the daily disaggregator will disaggregate the daily forcing down onto the model timestep.

For more information, see *l_daily_disagg*.

1.28.3 Major namelist changes

JULES vn4.0 also sees a major revamp of the science-related namelists. The monolithic JULES_SWITCHES namelist, and various others, are gone, and have been replaced with science section namelists. For more details, see *The JULES namelist files*.

This has been with the aim of providing a GUI for editing the JULES namelists using Rose, which is now available - see *Automatic upgrading and GUI using Rose*.

It also has the advantage that the new namelists are cut-and-paste-able between the UM and JULES, which should make it easier to ensure that the same science is being used in online and offline runs.

1.28.4 Removal of GNU make build files

After a period of supporting two build systems (FCM make and GNU make), it has been decided that support for GNU make should be removed. The overhead of maintaining two build systems was getting too large, and FCM make is preferred for several reasons:

Directory structure

- The directory level dependencies used by the JULES Makefile to ensure files are compiled in the correct order forced the directory structure to adapt to it.
- FCM make does automatic dependency analysis for each file to ensure they are compiled in the correct order, meaning the directory structure doesn't have to be compromised to keep the build system happy.

Dependencies

- The JULES GNU Makefiles required that dependencies be manually maintained, both in terms of the order of sub-makes and actual file dependencies within the sub-makes.
- FCM make automatically detects all dependencies and does things in the correct order.

Parallel builds

- JULES builds with GNU make could not be parallelised, because of the use of directory level sub-makes.
- FCM make considers each individual file, so builds can be parallelised.

Integration with Rose

- FCM make has good integration with Rose, allowing the Rose GUI for JULES to configure and run builds as well as the namelists.

1.28.5 Bugs and other changes

- Output for land points not comparing between land_only = T and F runs with 2D grid
- Incorrect behaviour when `spinup_end` == `data_end`
- Fixed overflow problem with `datetime_diff` when `datetime`s are too far apart
- Removed old implicit solver and ltimer code
- Unified management of printing and error reporting for UM and standalone

1.29 JULES version 3.4 Release Notes

n.b. A critical memory leak was found in JULES v3.4 that necessitated a new release, designated v3.4.1.

1.29.1 Changes to semantics of output

The output semantics used since JULES vn3.2 (i.e. state variables captured at the start of a timestep, flux variables captured at the end) were confusing some users. The semi-implicit scheme in JULES is designed so that the state and fluxes at the end of a timestep are consistent with each other, but under the previous semantics these were staggered by one timestep in output files.

All variables are now captured at the end of a timestep, so state and flux variables at a particular timestep in output files will be consistent with each other. A new option has been added to request the output of initial state, however very few users will have a use for this. It is still the case that the value in the `time` variable can be used to place snapshot data in time, and the values in `time_bounds` represent the interval over which a mean or accumulation applies.

More details can be found at *JULES output*.

1.29.2 Input and/or output of variables with multiple 'levels' dimensions has been improved

In previous versions of JULES since vn3.1, variables could only be input or output with a single 'levels' dimension. In particular, this caused problems with variables in the new snow scheme, which have two 'levels' dimensions on top of the grid dimensions (tiles and snow levels). This led to compromises being made with the snow layer variables:

- It was only possible to initialise the snow layer variables using a constant value, from a previous dump or using *`total_snow`*
- In output files, the snow layer variables were represented using a separate variable for each tile

This problem is solved in JULES vn3.4 - it is now possible to input and output variables with multiple 'levels' dimensions (there is not even a restriction to two 'levels' dimensions). This means that both compromises for snow layer variables detailed above have been removed.

1.29.3 Streamlined process for adding new variables for input and/or output

Although fairly simple, the process for adding a new variable for input and/or output in JULES vn3.1 - vn3.3 required several edits to be made, and hence provided many opportunities to make mistakes. This process is simplified in JULES vn3.4 to require fewer edits. More details can be found at *Implementing new variables for input and output*.

1.29.4 Other changes

Tidying of boundary layer code
Some small changes have been made to tidy up some of the boundary layer code (i.e. routines in `src/science/surface`) - this is mostly removing unused variables and tidying up subroutine argument lists.

OpenMP related changes
Some OpenMP directives have been added to certain loops. OpenMP is a form of shared-memory parallelism in which the user inserts directives (specially formatted code comments) providing information that allows the compiler to parallelise sections of code (in particular loops) without worrying about corrupting data. It is used in the UM, but is currently not enabled when compiling JULES standalone.

1.29.5 Bugs fixed

- Output (including dump files) not correctly generated for the last spin-up cycle when spin-up fails and *terminate_on_spinup_fail* = TRUE
- `lw_net` diagnostic does not include the contribution from the reflected incoming longwave if the emissivity is less than zero

1.30 JULES version 3.3 Release Notes

1.30.1 Ability to run JULES in parallel

JULES can now run multiple points in parallel, using multiple cores on the same machine or a cluster of machines. This is accomplished using MPI (Message Passing Interface), a standardised message passing interface. Several implementations of MPI are available, the most commonly used being MPICH2 and OpenMPI.

JULES takes advantage of the parallel I/O features in HDF5 / NetCDF4. These are not enabled by default, and so must be explicitly enabled when HDF5 / NetCDF4 are compiled. More information on how to do this can be found on the NetCDF website.

Information on how to build and run JULES in parallel can be found in the JULES User Guide. *Note that although this development has proven stable during testing, it is still experimental and is considered to be for advanced users only.*

1.30.2 Changes to documentation

From a users point of view, the most important change is that the JULES documentation and coding standards are now provided in two forms - HTML (this is the preferred format) and PDF. The HTML documentation is also available on the web at http://jules-lsm.github.io/.

This has been made possible by migrating the documentation from a single massive Word document to the Sphinx documentation generator (with some custom extensions to better support Fortran namelists). Although originally intended to document Python projects, Sphinx's extensibility has seen it adopted for a wide range of projects. Using Sphinx has several advantages over the previous monolithic Word document:

- Both forms of documentation (HTML and PDF) can be built from the same sources.
- The documentation is now split into several smaller files that are combined by Sphinx at build-time, leading to increased readability.
- reStructuredText, the markup language used by Sphinx, is a plain text format, meaning that it can be version controlled much more effectively than a Word document (which is treated by Subversion as a single binary entity).
- The only software required to update the documentation is your favourite text editor (rather than Word).

The JULES repository on PUMA has also been refactored so that configurations, documentation and examples sit in a separate project to the core Fortran code.

1.30.3 Other changes

Disambiguation of sea ice roughness lengths for heat and momentum
Prior to vn3.3, these were implicitly assumed to be equal by the code. They can now be set separately in the namelist `JULES_SURF_PARAM`.

Improvements to the numerics in the soil hydrology
Previously, the soil hydrology scheme coped poorly with significant gradients in soil moisture because of the sensitive dependence of the hydraulic conductivity and soil water suction on the soil moisture. See the new switch `l_dpsids_dsdz`.

Implicit numerics for land ice
Previously, the updating of land ice temperatures was always explicit, limiting the thickness of soil levels that can be used with standard time steps. There is now an option for implicit numerics for land ice - see the new switch `l_land_ice_imp`.

Scaling of land surface albedo to agree with a given input
An option has been added to prescribe the grid-box mean snow-free albedo to a given input (e.g. observations, climatology). See the new switch `l_albedo_obs`. For SW albedos, the albedos of the individual tiles are scaled linearly so that the grid-box mean albedo matches the observations, within limits for each tile. When VIS and NIR albedos are required then the input parameters are scaled and corrected in a similar manner. The change was included in the Global Land configuration at vn5.0: https://code.metoffice.gov.uk/trac/GL/ticket/8.

BVOC emissions now on a switch
Previously, BVOC emissions diagnostics were calculated all the time, regardless of whether they were output. A new switch - `l_bvoc_emis` - has been added to enable the calculation of these diagnostics only when required.

Improvements to logging
A new namelist file - `logging.nml` - has been added to give more control over log output from JULES. Previously all output was directed to `stdout`.

Specify namelist directory as an argument
It is now possible to specify the directory containing the namelist files as a command line argument to JULES. If no argument is given, JULES looks for the namelist files in the current working directory. Previously, JULES had to be executed in the directory containing the namelists - this change should make it easier to run JULES in batch mode.

1.30.4 Bugs fixed

- Initialisation of `chr1p5m` and `resfs` in `sf_exch`.
- Fix for potential divide-by-zero in `sf_stom` when running with `can_rad_mod = 1`.
- Various UM-related fixes not relevant to standalone JULES (ENDGAME, aerosol deposition scheme, etc.).

1.31 JULES version 3.2 Release Notes

JULES version 3.2 sees several enhancements and bug fixes in both the science and control code.

1.31.1 Standard Configurations

A set of standard science configurations have been defined. These are based on well tested operational Met Office models, and are intended to cover a wide range of use cases.

1.31.2 Improvements to output

In JULES version 3.1, under some circumstances, it was not entirely clear how the timestamps in output files applied to the values. This has been thoroughly addressed in version 3.2.

Changes have also been made to the attributes of output variables:

- The units attribute for output variables has been updated to be compliant with UDUNITS2.
- A CF conventions coordinates attribute has been added to all output variables that explicitly links the latitude and longitude to the data.

1.31.3 Biogenic Volatile Organic Compound (BVOC) emissions

Code written by Federica Pacifico for isoprene emissions has been implemented and extended to include monoterpene, acetone and methanol emissions. This addition is purely diagnostic in the standalone model (i.e. provides new output variables, but has no feedbacks), but will allow the UM to implement interactive BVOC emissions (i.e. with feedbacks) in the future.

A paper has been written describing and evaluating the isoprene emission scheme - Pacifico et. al., 2011. Atmos. Chem. and Phys., 11, 4371-4389 (PDF).

1.31.4 Alternative build system

It is now possible to build JULES using FCM make. FCM is a set of tools developed by the Met Office for managing and building source code, with a particular focus on making it easy to build large Fortran programs (such as JULES). FCM is open source software, and can be downloaded for free from Github.

1.31.5 Bugs fixed

- Array bounds error with `SICE_INDEX_NCAT`.
- Incorrect usage of `COR_MO_ITER`.
- Monthly/yearly output files not rolling over properly on certain configurations of GFortran.
- A collection of small memory leaks.
- Not able to read or write ASCII dumps with the new snow scheme on.
- Use fixed dimension names for output files (rather than using those given for input files).
- Using `can_rad_mod = 5` causes night-time dark respiration to be 0 under certain circumstances.

1.32 JULES version 3.1 Release Notes

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

1.32.1 Restructuring of the code

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

1.32.2 New I/O framework

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

1.32.3 User Interface changes

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

1.32.4 Other changes

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

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

1.33 JULES version 3.0 Release Notes

The major change in version 3.0 is the introduction of the 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:

- The pattern-scaling allows estimates of climate change for a broad range of emissions scenarios.
- New process understanding can be tested for its global implications.
- New process understanding can also be checked for stability before full inclusion in a GCM.
- 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.

1.34 JULES version 2.2 Release Notes

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 (UM). The UM uses preprocessor directives to compile different versions of routines depending on the selected science options. For compatibility with this system, JULES will now require a compiler with a preprocessor. This should not be noticed by the majority of users - most modern compilers include a preprocessor and the Makefile deals with setting up the appropriate preprocessor options.

Finally, JULES was added to the UM code repository as a mirror of the JULES repository at (UM version vn7.5, JULES vn2.2).

1.35 JULES version 2.1 Release Notes

Versions 2.1.1 and 2.1.2 were released to fix major bugs found in v2.1 - they contain no new features.

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.

1.35.1 Process descriptions

The main change is that a new multi-layer snow scheme is available. 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.

1.35.2 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, 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).

Finally, since JULESvn1.0 the MOSES and JULES code bases have been evolving separately, but with JULES v2.1 these differences have been reconciled with the UM.

1.36 JULES version 2.0 Release Notes

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.

1.37 JULES version 1.0

Initial public release of the JULES code. JULES v1.0 will only run for a single point and only supports ASCII data.

For further information about these releases, please see here.

CHAPTER

TWO

OVERVIEW OF JULES

This section provides a brief overview of JULES and an introduction to some of the key switches that determine what the model will simulate in a given run.

2.1 Overview

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 can be found at the JULES website and in the two JULES description papers (Best et al., 2011, GMD and Clark et al., 2011, GMD).

For both gridded and single point runs, JULES views each gridbox as consisting of a number of surface types. The fractional area of each surface type is either prescribed by the user or modelled by the TRIFFID sub-model. Each surface type is represented by a surface tile, and a separate energy balance is calculated for each surface tile. The gridbox average energy balance is found by weighting the values from each surface 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 surface 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. Each surface tile can be associated with its own soil tile, or all surface tiles can interact with a shared 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.

2.2 Some key switches

There are many variables that act together to determine how a run of JULES is set up and these are covered in detail in *The JULES namelist files*. Additionally, configurations illustrate suitable combinations of options. Here we highlight a few key switches that select broad areas of science, particularly for the benefit of new users.

The phenology model for natural vegetation can be enabled using `l_phenol` which uses the leaf turnover rate to calculate a time-varying Leaf Area Index (LAI).

To simulate carbon stocks in natural vegetation, the TRIFFID dynamic vegetation model can be enabled via the switch `l_triffid`. When TRIFFID is on, competition between tiles is switched on with `l_veg_compete` and the effect of nitrogen on vegetation growth is enabled via `l_nitrogen`.

The crop model, which is simulates phenology and carbon stocks in crops, can be switched on by setting the number of crop tiles `ncpft` to a non-zero value.

The crop model and TRIFFID cannot currently be used together. To simulate agricultural areas within TRIFFID, a fraction of the gridbox can be reserved for agricultural Plant Functional Types (PFTs) (as defined by *crop_io* > 0). Agricultural PFT competition and a representation of harvest carbon can be switched on with *l_trif_crop*.

If neither the phenology model nor the crop model are used, LAI for each vegetation tile can be set to a constant (*lai_io*) or a time series or seasonal cycle can be prescribed (*JULES_PRESCRIBED*).

To simulate carbon and nitrogen stocks in the soil, the 4-pool model should be selected by setting *soil_bgc_model* = 2. This option adds prognostic soil pools and must be used with the TRIFFID vegetation model. If TRIFFID is not used, prescribed soil pools must be invoked via *soil_bgc_model* = 1. Layered soil pools are used if *l_layeredc* = .TRUE..

A multi-layer snow model can be selected using *nsmax*. Parameterisations of surface and subsurface runoff generation are controlled using *l_top* and *l_pdm*, while the routing of water in rivers uses *l_rivers*.

CHAPTER

THREE

BUILDING AND RUNNING JULES

This section details the options available for compiling and running JULES.

3.1 Considerations

Depending on your use case, there are two main things that you need to consider:

3.1.1 Do I need NetCDF?

NetCDF is a data format (and associated software libraries) specifically designed for large-scale scientific data. It has two major benefits over raw binary data:

1. It is machine-independent, so endianness is not an issue when moving datasets between machines
2. It is self-describing, so as well as containing raw data, NetCDF files also contain metadata describing the data (e.g. variable names, units, origin). Many tools are capable of exploiting this metadata to simplify processing.

JULES can be built with or without NetCDF, however *building JULES without NetCDF limits the functionality of JULES*. Without NetCDF, JULES will use a dummy NetCDF library which allows the program to build but provides no functionality. Any attempt to use NetCDF files as input with this option will result in a runtime error. All input files must be columnar ASCII, meaning that the user is restricted to running at a single point only. Output files will automatically use a columnar ASCII format with headers. File formats are discussed in more detail in *Input files for JULES*.

3.1.2 Do I need parallel processing?

Note: For running JULES at a single point, parallel processing provides no advantage. However, if JULES is already compiled with OpenMP or MPI enabled, it is still possible to run a single point by simply specifying the number of OpenMP threads and/or MPI tasks to be 1.

JULES is capable of exploiting parallel processing techniques to reduce processing time for distributed/gridded simulations. There are two different methods JULES can use:

OpenMP

OpenMP is a form of compiler-assisted parallelisation that uses directives for shared-memory, loop-level parallelism across multiple cores on a machine (OpenMP is *not* capable of utilising a cluster of machines).

This form of parallelism is not as effective as MPI, but may provide some speedup and does not require a specially compiled NetCDF library.

MPI

MPI (Message Passing Interface) is a standardised message passing interface. MPI coordinates the running of multiple 'tasks' in parallel, potentially on several machines (or nodes), and provides mechanisms for these tasks to communicate with each other.

JULES takes advantage of the parallel I/O features available in HDF5 and NetCDF4, which enable multiple MPI tasks to read from and write to the same NetCDF file(s) at the same time. These features must be explicitly enabled when NetCDF is compiled (see *Required software*).

It is also possible to use MPI and OpenMP together, where each MPI task has a number of OpenMP threads, however this is very advanced and beyond the scope of this document.

3.2 Required software

Building a JULES executable requires FCM and one of the supported Fortran compilers (see *Building JULES using FCM*). The Fortran 90 NetCDF interface library is required to use gridded data (i.e. data for more than a single location).

To be able to automatically upgrade namelists between JULES versions or use a GUI to configure JULES runs, Rose is required.

All of this software is freely available:

- GFortran, the GNU GCC Fortran compiler - http://www.gnu.org/software/gcc/fortran
- FCM - http://metomi.github.io/fcm/doc
- Rose - http://metomi.github.io/rose/doc/html/index.html
- NetCDF libraries - http://www.unidata.ucar.edu/software/netcdf

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

3.2.1 Building JULES with NetCDF

To build JULES with NetCDF, it must be told where to find the NetCDF library files. JULES needs two pieces of information - the directory containing the NetCDF archive files, `netcdf.a` and `netcdff.a` (the *'NetCDF library path'*), and the directory containing the NetCDF Fortran 90 module file, `netcdf.mod` (the *'NetCDF include path'*). In a standard NetCDF install, these are often `/usr/lib` and `/usr/include` or `/usr/local/lib` and `/usr/local/include` respectively.

If the `nc-config` program is installed on your system (run `which nc-config` to find out), this can be used to determine values for the NetCDF library path (`nc-config --flibs`) and NetCDF include path (`nc-config --includedir`). When JULES is built with NetCDF, users can supply either ASCII or NetCDF input files, and all output will be NetCDF.

3.2.2 Building and running JULES with MPI

Warning: For advanced users only

In order to build and run JULES with MPI, additional software is required:

1. An implementation of MPI compiled using the same compiler you will be using to compile JULES. Several implementations of MPI are available, the most commonly used being MPICH2 and OpenMPI.

   Note: The `bin` directory of your MPI installation must be in your `$PATH`

2. A version of HDF5/NetCDF4 compiled *with parallel I/O enabled*, using the MPI implementation installed above. This is *not* the default way to compile NetCDF, and must be explicitly enabled. More information on how to do this can be found on the NetCDF website.

3.3 Building JULES using FCM

FCM is a code management and build system developed by the Met Office with a particular focus on simplifying the process of building large Fortran programs. In this section, we will be using the build tool - FCM make.

As part of the build process, FCM make will analyse the dependencies of every Fortran file and automatically compile them in the correct order.

FCM make must be given a configuration file that it uses to determine how to build the source code. Extensive documentation on FCM make configuration files is available online.

Help pages for the FCM make command itself (rather than the configuration file) can be accessed using the command:

```
fcm help make
```

The FCM configuration file for building JULES is `etc/fcm-make/make.cfg`. This file uses the environment variables below to determine the settings to use when compiling JULES.

Running FCM make with this configuration file will create some files and directories in the specified build directory (see the `-C` option of `fcm make`; defaults to the current working directory). The JULES executable will be produced in the specified build directory at `build/bin/jules.exe`.

3.3.1 Environment variables used when building JULES using FCM make

`JULES_PLATFORM`

Used to select settings for a pre-defined platform. The default values of other variables may depend on the choice of this setting; differences from the generic defaults are included in the descriptions below.

Note: If you have many users using the same platform to run JULES, you may want to contribute a suitable platform configuration.

Permitted value	Purpose
`custom`	**Default.** Use a custom configuration entirely determined by the other environment variables. The default values of those variables are set in this platform's configuration file.
`vm`	Use settings for the *JULES development virtual machine*.
`ceh`	Use settings for the GFortran compiler on the CEH Linux systems.
`jasmin-lotus-intel`	Use settings for the Intel compiler on the Lotus system at JASMIN.
`jasmin-gcc-nompi`	Use settings for the gfortran compiler on the JASMIN Cylc server.
`jasmin-intel-nompi`	Use settings for the intel compiler on the JASMIN Cylc server.
`meto-linux-gfortran`	Use settings for the GFortran compiler on Met Office Linux systems.
`meto-linux-nagfor`	Use settings for the NAG compiler on Met Office Linux systems. **Warning:** This build configuration is intended for correctness checking only, not production runs.
`meto-linux-intel-nompi`	Use settings for the Intel compiler *without* MPI on Met Office Linux systems.
`meto-linux-intel-mpi`	Use settings for the Intel compiler *with* MPI on Met Office Linux systems.
`meto-xc40-cce`	Use settings for the Cray Compiler Environment on the Met Office Cray XC40 system.
`uoe-linux-gfortran`	Use settings for the GFortran compiler on University of Exeter Linux system (SL7).

`JULES_REMOTE`, `JULES_REMOTE_HOST`, `JULES_REMOTE_PATH`

Warning: Advanced users only

Used to determine whether the build will happen on a local or remote machine.

Permitted value	Purpose
`local`	**Default.** All compilation occurs on the local machine.
`remote`	Code is extracted on the local machine and mirrored to `${JULES_REMOTE_HOST}@${JULES_REMOTE_PATH}`, where `JULES_REMOTE_HOST` is the name of the remote machine and `JULES_REMOTE_PATH` is the path on the remote machine. The compilation can then be completed on the remote machine. See below for an example.

`JULES_COMPILER`

Used to select compiler specific settings.

Permitted value	Purpose
`gfortran`	**Default.** Use settings for the GNU Fortran compiler.
`intel`	Use settings for the Intel Fortran compiler.
`nagfor`	Use settings for the NAG Fortran compiler.
`cray`	Use settings for the Cray Compiler Environment.

`JULES_BUILD`

Used to select the type of build.

Permitted value	Purpose
`normal`	**Default.** Compile JULES normally.
`debug`	Compile JULES with additional settings for debugging.
`fast`	Compile JULES with additional settings for faster execution.

`JULES_OMP`

Used to determine whether to build with OpenMP or not.

Permitted value	Purpose
`noomp`	**Default.** Compile JULES with OpenMP off.
`omp`	Compile JULES with OpenMP on.

`JULES_MPI`

Used to determine whether to build with MPI enabled or not.

Permitted value	Purpose
`nompi`	**Default.** Compile JULES without MPI support.
`mpi`	Compile JULES with MPI support.

`JULES_NETCDF`

Indicates whether to use a dummy NetCDF library or a 'real' NetCDF library.

Permitted value	Purpose
`nonetcdf`	**Default.** Use a dummy NetCDF library.
`netcdf`	Use a 'real' NetCDF library. The NetCDF installation to use is specified using one of: • `JULES_NETCDF_PATH` • `JULES_NETCDF_INC_PATH` and `JULES_NETCDF_LIB_PATH`

JULES_NETCDF_PATH
Path to NetCDF installation.

This sets `JULES_NETCDF_INC_PATH = $JULES_NETCDF_PATH/include` and `JULES_NETCDF_LIB_PATH = $JULES_NETCDF_PATH/lib`. These can be overridden by setting the variables directly.

JULES_NETCDF_INC_PATH
Path to NetCDF include directory (i.e. directory containing `netcdf.mod`).

JULES_NETCDF_LIB_PATH
Path to NetCDF library directory (i.e. directory containing `libnetcdff.a` and `libnetcdf.a`).

Note: When compiled in parallel mode, NetCDF must be statically linked. This means the compiler must be able to find all required library and include files (i.e. for NetCDF, HDF5, curl and zlib) in `JULES_NETCDF_INC_PATH`, `JULES_NETCDF_LIB_PATH` or the default search path.

JULES_FFLAGS_EXTRA
Any additional compiler flags you wish to add to the build. For example, to activate additional compiler checks.

JULES_LDFLAGS_EXTRA
Any additional library flags you wish to add to the build. This may need to include both the linker flags themselves and, if you are linking in a new library, the flags specifying the path to the new library object.

Note: When adding a completely new external dependency it is likely you will need to edit or override the FCM make build configuration files. The FCM make tool performs a dependency analysis on the JULES source tree to ensure all of the required files are present. Any new external sources must be added to the list of exclusions from this analysis or the build will fail when the external files cannot be found in the JULES working copy.

JULES_SOURCE
The full path to the copy of JULES being compiled. This could be a directory path or an FCM/Subversion/file URL to a repository location. This variable is used by the configuration file contained in many Rose fcm_make apps, but is not read by JULES itself.

3.3.2 Example FCM make commands

To create a normal JULES executable without NetCDF using the GFortran compiler (taking advantage of the default values for the environment variables):

```
$ fcm make -j 2 -f etc/fcm-make/make.cfg --new
```

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

```
$ export JULES_COMPILER=intel
$ export JULES_BUILD=fast
$ export JULES_NETCDF=netcdf
$ export JULES_NETCDF_PATH=/path/to/netcdf  # Replace this with the correct path
$ fcm make -j 2 -f etc/fcm-make/make.cfg --new
```

To create a fast JULES executable with NetCDF using the GFortran compiler on a Met Office Linux system (making use of the platform setting):

```
$ export JULES_PLATFORM=meto-linux-gfortran
$ export JULES_BUILD=fast
$ export JULES_NETCDF=netcdf  # Note that we don't need to specify paths
$ fcm make -j 2 -f etc/fcm-make/make.cfg --new
```

To create a normal JULES executable with NetCDF and OpenMP using the Intel compiler on a remote machine:

```
localhost $ export JULES_REMOTE=remote
localhost $ export JULES_REMOTE_HOST=my-host
localhost $ export JULES_REMOTE_PATH=/path/on/remote/host
localhost $ export JULES_COMPILER=intel
localhost $ export JULES_OMP=omp
localhost $ export JULES_NETCDF=netcdf
localhost $ export JULES_NETCDF_PATH=/path/to/netcdf  # Replace this with the path ON␣
↪THE REMOTE MACHINE
localhost $ fcm make -f etc/fcm-make/make.cfg --new  # This does the extract and␣
↪mirror steps
localhost $ ssh -Y my-host
my-host $ cd /path/on/remote/host
my-host $ fcm make -j 4 --new  # This does the preprocess and build steps
```

To create a normal JULES executable with MPI enabled, using the Intel compiler with array bounds checking turned on:

```
$ export JULES_COMPILER=intel
$ export JULES_MPI=mpi
$ export JULES_NETCDF=netcdf  # We have to use NetCDF for distributed simulations
$ export JULES_NETCDF_PATH=/path/to/parallel/netcdf  # NetCDF must be compiled with␣
↪parallel I/O enabled
$ export JULES_FFLAGS_EXTRA="-check bounds"  # Must be quoted because of the space
$ fcm make -j 2 -f etc/fcm-make/make.cfg --new
```

3.3.3 Tips for effective use of FCM make

- To check the current values of the environment variables JULES will use to build, use the command `env | grep JULES`
- If you always use the same compilation options for JULES, consider adding the export lines to the `.profile` file in your `$HOME` directory. Commands in the `.profile` file are automatically executed in any shell that you open, so defining environment variables there ensures your build environment remains consistent across shells and restarts of your computer. The definitions can still be overridden on the command line if required.

3.4 Running JULES

The user interface of JULES consists of several files with the extension `.nml` containing Fortran namelists. These files and the namelist members are documented in more detail in *The JULES namelist files*. These namelists are grouped together in a single directory. That directory is referred to as the *namelist directory* for a JULES run. In most use cases, this is practically abstracted away by the use of the rose/cylc workflow. This provides a GUI and rich ecosystem for integration of JULES into a larger workflow (eg compile-run-analyse).

Once a *JULES executable is compiled* and the *namelists* are set up, JULES can be run in one of two ways:

1. Run the JULES executable in the namelist directory with no arguments:

   ```
   cd /path/to/namelist/dir
   /path/to/jules.exe
   ```

2. Run the JULES executable with the namelist directory as an argument:

   ```
   /path/to/jules.exe  /path/to/namelist/dir
   ```

Warning: Any relative paths given to JULES via the namelists (e.g. *file* in *JULES_FRAC*) will be interpreted *relative to the current working directory*.

This means that if the user plans to use the second method to run JULES (e.g. in a batch environment), it is advisable to use fully-qualified path names for all files specified in the namelists.

To allow runs to be portable across different machines, it is common to specify data files relative to the namelist directory. In this case, JULES must be run using the first method to allow the relative paths to be resolved correctly.

3.4.1 General example of running JULES from the command line

1. Move into the JULES root directory (the directory containing `includes`, `src` etc.):

   ```
   $ cd /jules/root/dir
   ```

2. Build JULES:

   ```
   $ fcm make -f etc/fcm-make/make.cfg
   ```

3. Move into the namelist directory:

   ```
   $ cd /path/to/namelist/dir
   ```

4. Run the JULES executable:

   ```
   $ /path/to/jules.exe
   ```

3.4.2 Running JULES with OpenMP

If JULES is compiled with OpenMP, then it must be told how many OpenMP threads to use. This is done using the environment variable `OMP_NUM_THREADS`:

```
$ export OMP_NUM_THREADS=4  # Use 4 threads for OpenMP parallel regions
$ /path/to/jules.exe
```

3.4.3 Running JULES with MPI

When running JULES using MPI, JULES attempts to find a suitable decomposition of the grid depending on how many MPI tasks are made available to it. Each MPI task can then be thought of as its own independent version of JULES, with each task being responsible for a portion of the grid. Each task reads its portion of the input file(s), performs calculations on those points and outputs its portion of the output file(s). Tasks only communicate in order to read and write dump files - this ensures that dump files are consistent regardless of decomposition, i.e. a dump from any run (MPI or not; different numbers of MPI tasks), can be used to (re-)start any other run and produce identical results, providing the overall model grids are the same.

None of the namelists or namelist members are parallel-specific - the same *JULES namelists* can be used to run JULES with or without MPI, and the final results will be identical.

If JULES is compiled with MPI, then it must be run using commands from your MPI distribution (usually called `mpiexec` and/or `mpirun`):

```
$ mpirun -n 4 /path/to/jules.exe  # Run JULES using 4 MPI tasks
```

Detailed discussion of `mpiexec/mpirun` is beyond the scope of this document - please refer to the documentation for your chosen MPI distribution for the available options and features.

3.5 Automatic upgrading and GUI using Rose

Rose is a collection of tools for managing the building and running of scientific applications.

See also:

Please familiarise yourself with the Rose documentation before continuing with this section.

Note: This section assumes Rose is installed.

We will not be using Rose Bush or Rosie, so those components need not be installed.

It is not necessary to install Cylc, but some functionality will not be available. This will be noted as we go.

JULES uses Rose primarily to provide a graphical interface for configuring and running JULES, but also to allow automatic upgrading of JULES runs from one version to the next.

A Rose suite for JULES will normally contain two applications - an `fcm_make` application for building JULES and a `jules` application for configuring the namelists and running JULES.

3.5.1 Creating a Rose suite from existing namelists

To enable users to quickly transition to Rose and the extra functionality it provides, a tool is distributed with JULES that can convert existing namelists to a Rose suite.

To convert vn3.4 namelists to a vn4.7 Rose suite, run the following command in the directory containing the namelists:

```
create_rose_app vn3.4 vn4.7 namelist_path suite_name jules_dir
```

Where `jules_dir` is the path to the root directory of the most recent JULES code release on your machine.

The `namelist_path` can be the full or a relative path.

This will create a directory called `suite_name` in ~/roses/ directory which contains a fully functional Rose suite.

To convert namelists to a Rose suite without upgrading the version, just give the same version for both.

3.5.2 Using Rose to upgrade existing namelists

It is not necessary to use Rose to configure and run JULES - Rose can be used just to upgrade existing namelists (at vn3.4 or later).

In order to use Rose to upgrade existing namelists from vn3.4 to vn4.0, just execute the following commands in the directory containing your namelists:

```
# Creates a Rose suite at rose-suite
$JULES_ROOT/bin/create_rose_app vn3.4 vn4.0

# Remove the current namelists
rm -rf *.nml

# Use Rose to generate the new namelists
rose app-run -i -C rose-suite/app/jules

# Remove the Rose suite and other generated files
rm -rf rose-suite .rose-config_processors-file.db rose-app-run.conf
```

3.5.3 Upgrading an existing JULES Rose suite

Upgrading an existing JULES Rose suite is even more simple than upgrading the namelist files directly. To see the versions it is possible to upgrade to, run the command:

```
  rose app-upgrade -M $JULES_ROOT/rose-meta -C /path/to/rose/suite/app/jules --all-
↪versions
&& rose macro --fix -C app/jules

  rose app-upgrade -M $JULES_ROOT/rose-meta -C /path/to/rose/suite/app/fcm_make --all-
↪versions
&& rose macro --fix -C app/fcm_make
```

To then upgrade to one of those versions, the command is:

```
  rose app-upgrade -M $JULES_ROOT/rose-meta -C /path/to/rose/suite/app/jules <version>
&& rose macro --fix -C app/jules

  rose app-upgrade -M $JULES_ROOT/rose-meta -C /path/to/rose/suite/app/fcm_make
↪<version>
&& rose macro --fix -C app/fcm_make
```

3.5.4 Configuring JULES with a graphical interface

Using a Rose suite to run JULES has the advantage that it can be configured graphically using Rose Config Edit.

To launch the graphical editor, the following command is used:

```
# To edit the whole suite, including build configuration
rose config-edit -M $JULES_ROOT/rose-meta -C /path/to/rose/suite &

# To edit just the namelists
rose config-edit -M $JULES_ROOT/rose-meta -C /path/to/rose/suite/app/jules &
```

where `$JULES_ROOT` is the root directory of your JULES installation. For more information on using the config editor, see the Rose documentation

Clicking on a variable name in the editor opens the corresponding page in this documentation.

3.5.5 Running a JULES Rose suite

Without Cylc

To run JULES from a Rose suite without Cylc, we just use Rose to generate the namelists. JULES is then built and run as normal - see *Building and running JULES*.

To generate namelists in the current directory from a Rose suite at `/path/to/rose/suite`, use the following command:

```
rose app-run -i -C /path/to/rose/suite/app/jules
```

With Cylc

Warning: This requires Cylc to be installed and configured.

Once a JULES Rose suite has been suitably configured using the graphical editor, it can be run using the following command:

```
rose suite-run -C /path/to/rose/suite
```

This will set the suite running, and will launch the Cylc GUI to allow you to see the status of your suite as it runs. The GUI also allows you to view log files etc. - these can be useful when a job fails!

CHAPTER

FOUR

INPUT FILES FOR JULES

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

4.1 General principles

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

In JULES, the input grid is comprised of the following information:

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

The input grid is specified by the user in the namelist *`JULES_INPUT_GRID`*. The model grid is then constructed by selecting the desired points from this input grid, the default being that only the land points in the input grid will be processed. All output is on the model grid.

Note: All input data must use the same grid, including any ancillaries and initial conditions.

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

ASCII files
: `.asc`, `.txt` and `.dat`

NetCDF files
: `.nc` and `.cdf`

4.2 ASCII files

JULES only supports the use of ASCII files for data at a single location. In this case, the input grid can be specified either as a 1D grid with length 1 or as a 2D grid of size 1 x 1. The data should be laid out in columns with one timestep of data per row (with time increasing with the number of rows). For variables with additional 'levels' dimensions (e.g. soil layers), the values for each level should be in consecutive columns.

Note: Variables should be given to JULES in the order they appear in the file, and there should be no unused variables in between. This may mean that some datasets may require pre-processing for use with JULES, even if they are already columnar.

If the first character of a line is either # or !, the line is taken to be a comment. JULES reads no information from comments - they are purely for annotating the dataset for users.

4.2.1 Example ASCII input

ASCII meteorological forcing data

```
# Meteorological data for Loobos, 1997.
# One year of 30 minute data.
#   Down  Down       Rainfall       Snowfall      Air       Wind                  Specific
#   SWR   LWR         rate            rate       temp.      speed     Pressure    humidity
#(W m-2) (W m-2)  (kg m-2 s-1) (kg m-2 s-1)   (K)        (m s-1) )   (Pa)        (kg kg-
↪1)
    0.0   187.8     0.000E+00     0.000E+00   259.800     2.017     102400.0    1.384E-
↪03
    0.0   186.9     0.000E+00     0.000E+00   259.700     3.770     102400.0    1.384E-
↪03
    0.0   186.7     0.000E+00     0.000E+00   259.600     4.290     102400.0    1.373E-
↪03
# ...
```

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

Initial conditions

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

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

Time varying data with an additional dimension

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

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

4.3 NetCDF files

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

- All input files use the same grid.
- All input files use the same dimension names (for grid dimensions, any additional dimensions and the time dimension).
- The dimensions for each variable appear in the correct order - `(points, z1, z2, ..., t)` for a 1D grid and `(x, y, z1, z2, ..., t)` for a 2D grid, where the `z1, z2, ...` (levels) and `t` (time) dimensions are only present when the variable and context in which the variable is being used require them.
- If using NetCDF for data at a single location, the grid dimensions are still expected to exist with size 1.

4.4 File name templating

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

Substitution strings are 3-character strings, starting with `%`. JULES will automatically detect the use of either form of templating by checking for the presence of the substitution strings in file names.

4.4.1 Time templating

If any of the time templating substitution strings are present in a file name, then JULES assumes time-templating is to be used. The valid substitution strings for time templating are:

Substitution string	Replaced with
`%y4`	4-digit year
`%y2`	2-digit year
`%m2`	2-digit month
`%m1`	1- or 2-digit month
`%mc`	3-character month abbreviation
`%d2`	2-digit day of month

JULES will automatically detect the period (or frequency) of files based on the specific substitution strings in the following manner:

This means that monthly files must also have a year substitution string present, and daily files must have both month and year substitution strings present. Only yearly, monthly and daily files are allowed with time templating, with each file containing a single period (year, month or day respectively) of data. For yearly files, the first data in each

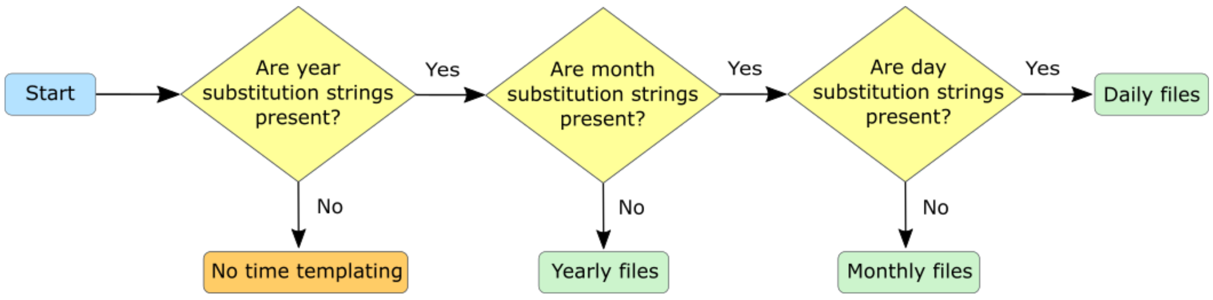
Start
Are year substitution strings present?
Yes
No
No time templating
Are month substitution strings present?
Yes
No
Yearly files
Are day substitution strings present?
Yes
No
Monthly files
Daily files

Fig. 1: Flow diagram showing detection of file period from time templated string

file must apply from 00:00:00 on 1st January for each year. For monthly files, the first data in the file must apply from 00:00:00 on the 1st of the month. For daily files, the first data in the file must apply from 00:00:00 on the given day. Other configurations can be specified using a list of files with their respective start times.

4.4.2 Variable name templating

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

4.4.3 Examples of file name templating

Time templating only

Data is in monthly files with all related variables in the same file.

Template:

```
/data/met_data_%y4%m2.nc
```

Example filenames:

```
/data/met_data_199001.nc
/data/met_data_199002.nc
...
/data/met_data_200410.nc
```

Variable name templating only

Ancillary (non-time-varying) data with each variable in similarly named but separate files.

Template:

```
/ancil/soil_%vv.nc
```

Example filenames:

```
/data/soil_satcon.nc
/data/soil_sathh.nc
```

Time and variable name templating together

Data is in monthly files with each variable in similarly named but separate files.

Template:

```
/data/%vv_%y4%mc.nc
```

Example filenames:

```
/data/Rain_1990jan.nc
/data/Wind_1990jan.nc
...
/data/Rain_2000oct.nc
/data/Wind_2000oct.nc
```

Variable name templating with a list of files

Data in 6-monthly files with each variable in similarly named but separate files.

Since the time templating cannot handle 6-monthly files, the files and their start times must be specified as a list. However, variable name templating can still be used.

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

Template list:

```
./%vv/met_%vv_199001.nc
./%vv/met_%vv_199007.nc
...
./%vv/met_%vv_199801.nc
```

Example filenames:

```
./Rain/met_Rain_199001.nc
./Wind/met_Wind_199001.nc
./Rain/met_Rain_199007.nc
./Wind/met_Wind_199007.nc
...
./Rain/met_Rain_199801.nc
./Wind/met_Wind_199801.nc
```

4.5 Temporal interpolation

Time-varying data as inputs into JULES are provided in two types - instantaneous states (e.g. air temperature, surface pressure, lai) or fluxes (e.g. radiation, precipitation). Because the data are on discrete timesteps, the value of an instantaneous variable applies at the timestamp (e.g. air temperature at 0800). However, values of the fluxes represent averages over the data timestep (e.g. 3-hour average rates). Different datasets supply the data as averages over the previous data timestep (backwards average) or the next data timestep (forwards average).

In order for the numerics to remain stable, it is recommended to run JULES with a model timestep of 1 hour or shorter. If the data timestep is longer than the model timestep, interpolation is required. How interpolation is performed for a particular variable depends on whether the variable is an instantaneous state or a flux.

4.5.1 Interpolation flags

When JULES needs to know what type of interpolation to use for a variable, the following flags are used.

`i`

Linear interpolation from the data timestep to the model timestep.

For instantaneous data (e.g. air temperature, surface pressure), this is almost always the flag that should be used.

`nb`, `nc` and `nf`

Values will be held constant with time for all model timesteps associated with a particular data timestep.

One of these flags should be used for flux variables that are *discontinuous* by nature, e.g. precipitation.

`nb` should be used if the dataset uses backwards average values, `nf` should be used if the data set uses forwards average values and `nc` should be used if the dataset uses centred average values (this is quite rare).

`b`, `c` and `f`

Data is interpolated using a simplified version of the Sheng and Zwiers (1998)[1] method that conserves the period means of the data.

One of these flags should be used for flux variables that are *continuous* in nature, e.g. radiation.

In order to ensure conservation of the average, these flags should be used only if the data period is an even multiple of the model timestep (i.e., if `data_period = 2 * n * timestep_len; n = 1, 2, 3, ...`). The curve-fitting process tends to produce occasional values near turning points that fall outside the range of the input values.

Similar to above, `b` should be used if the dataset uses backwards average values, `f` should be used if the data set uses forwards average values and `c` should be used if the dataset uses centred average values.

In order to perform interpolation, JULES may require input data for one or two data timesteps that fall before or after the times for the integration:

Flag	Extra data timesteps required
`nf`	Only requires data that falls within the integration times
`i`, `nb`, `nc`	Requires one data timestep beyond the end of the integration
`nb`	Requires two data timesteps beyond the end of the integration
`nf`	Requires one data timestep before the start and one data timestep beyond the end of the integration
`nc`	Requires one data timestep before the start and two data timesteps beyond the end of the integration

Also, note that for centred data (flags `c` and `nc`) the time of the data should be given as that at the start of the averaging period, rather than the centre, e.g. the 3-hour average over 06H to 09H, centred at 07:30H, should be treated as having timestamp 06H.

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

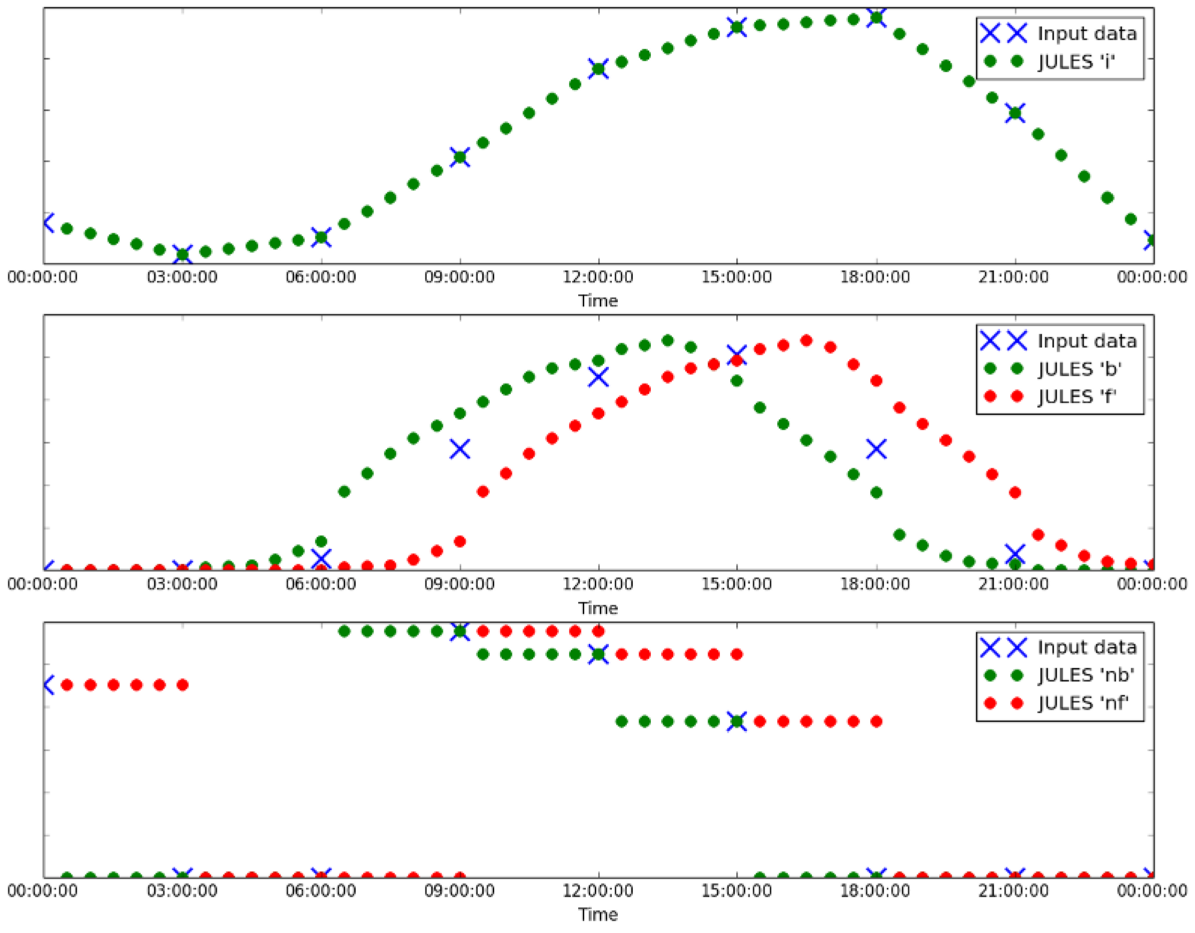

Fig. 2: Examples of data interpolated with `i`, `nb`, `nf`, `b` and `f`, plotted against the data they are derived from

CHAPTER

FIVE

JULES OUTPUT

JULES separates output into one or more output 'profiles'. Within each profile, all variables selected for output are written to the same file with the same frequency (also referred to as the 'output period'). The output period can be any multiple of the model timestep, including calendar months or years.

Most output is provided on the model grid only. Some variables are provided on the river routing model grid instead. Each output profile can contain **either** model grid **or** river routing model grid variables, but not both.

Each output file contains the latitude and longitude of each point to allow the points to be located in a grid if desired (e.g. for visualisation). Output files also contain two time related variables to locate the values in time (this is described in more detail *below*).

JULES is capable of performing five different types of time-processing - snapshot (instantaneous) values, time averages, time minima, time maxima and time accumulations. Snapshots are instantaneous values produced during the first model timestep of each output period. Time averages, minima, maxima and accumulations are calculated over the output period. Each output variable is annotated with a CF convention `cell_methods` attribute to indicate whether it is a snapshot value (`time : point`), time average (`time : mean`), time minimum (`time : minimum`), time maximum (`time : maximum`) or time accumulation (`time : sum`).

Each profile can be considered as a separate data stream. By using more than one profile the user can, for example:

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

All output files will be NetCDF if JULES is compiled with 'proper' NetCDF libraries (see *Building and running JULES*). Otherwise all output will be in columnar ASCII files.

5.1 Associating output values with the correct time

JULES output files contain two time related variables to allow model output to be associated with the correct model time:

`time`

For each output period, this variable contains the time of the end of the output period. This is the time that any snapshot values apply at.

`time_bounds`

For each output period, this variable contains two values - the start and end of the output period. The output period is then the half-open interval given by:

```
time_bounds(1) < time <= time_bounds(2)
```

This is the interval that means, minima, maxima and accumulations are calculated over.

During each model timestep, JULES captures values for output at the end of the timestep (i.e. after all the science code). This means that in output files, snapshot data at a particular timestep is:

- The state of the model at the end of the model timestep.

- The fluxes that produced that state over the model timestep.

Due to the way the model equations work, this ensures that all output at a given timestep in the output files is consistent.

5.2 Initial data

With the formulation given above, the initial state of the model (i.e. the state at the beginning of the first timestep of a section) is never output (except to dump files). For the majority of users, this will not be an issue. If the initial state is required, it is possible for an output profile to output the initial state for each section of a run (i.e. initial state of each spinup cycle and the main run) to a separate file - see `output_initial`.

Warning: In initial data files, **only snapshot values for state variables will be valid**. All other variables specified in the output profile will exist in the file, but their values will be garbage - *not necessarily NAN* - so use these files with caution.

5.3 Dump files

JULES writes dump files (a snapshot of the current model state) at several points during a run. These can be used to restart the model from that point if desired. The times that dump files are written are:

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

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

Prior to vn4.3, the dump file contained sufficient prognostic variables such that the pre-dump model state could be recreated. From vn 4.3 onwards, the dump file now includes ancillary data. The model can optionally restart from these data rather than the values given in the ancillaries namelists. Latitude and longitude information are also now written to (but not read from) the dump file to aid users wishing to interrogate dump files for debugging or other purposes.

CHAPTER

SIX

THE JULES NAMELIST FILES

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

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

These files have specific names, and JULES expects all these files to exist for every run (even when their contents are not required). JULES also expects that the namelists within each file appear in the order given below.

6.1 Introduction to Fortran namelists

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

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

```
&GROUP_NAME
  char_variable = "a char variable",
  logical_variable = T,
  nitems = 5,
  list_variable = 0.1  0.2  0.3  0.4  0.5
/
```

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

Values for character variables must be enclosed in either single(' ') or double (" ") quotes. Logical values can be specified using `.TRUE./.FALSE.` or with the shorthand `T/F`. Integer and real values are specified simply by giving the value. The vast majority of compilers (all tested compilers) allow lists to be specified either horizontally or vertically, depending on preference. The following definitions are identical:

```
list_variable = 0.1  0.2  0.3  0.4  0.5

list_variable(1) = 0.1
list_variable(2) = 0.2
list_variable(3) = 0.3
```

(continues on next page)

(continued from previous page)

```
list_variable(4) = 0.4
list_variable(5) = 0.5
```

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

```
&GROUP_NAME
  char_variable = "a char variable",
  nitems = 5,
  list_variable = 0.1  0.2  0.3  0.4  0.5
/
```

6.2 `jules_prnt_control.nml`

This file contains one namelist called *JULES_PRNT_CONTROL*.

This namelist sets options for output of diagnostic and informative messages.

6.2.1 `JULES_PRNT_CONTROL` namelist members

`JULES_PRNT_CONTROL::`**`prnt_writers`**

Type
integer

Permitted
1,2

Default
1

Selects which tasks in a parallel job will write informative output.

1	All tasks write output
2	Only the first task (Task 0) writes output

6.3 `jules_surface_types.nml`

This file configures the surface types used by JULES. It contains one namelist called *JULES_SURFACE_TYPES*.

The surface type IDs, which were introduced in the UM in order to identify the surface types present in input and output, have been made available to standalone. The defined surface type IDs are given in the description here in brackets (#). In order to keep the GUI from appearing cluttered, the surface types have been added with `compulsory=false`, unless they are attached to specific science options allowing them to be triggered off, which would be the preferred method. A `compulsory=false` surface type, can be added and removed in the GUI window as described in the table below. To:

Add	"Add latent variable" using the right click menu, which opens a list of defined surface types.
Re-move	"Remove" the variable using the cog menu.

Note: Please be aware that while the surface type IDs have been made available and are used to check the surface type configuration at runtime, they are not yet used by the JULES I/O.

6.3.1 `JULES_SURFACE_TYPES` namelist members

Note: The total number of surface types to be modelled is called `ntype`, and is given by `ntype = npft + nnvg`.

In the original 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 normally set in *JULES_FRAC*.

If the crop model is active (i.e. `ncpft` > 0), then `nnpft = npft - ncpft` where `nnpft` is the number of natural PFTs.

Vegetation surfaces must always be present first in any list of surfaces.

`JULES_SURFACE_TYPES::`**`npft`**

Type
integer

Permitted
>= 1

Default
-32768

The number of plant functional types (PFTs) to be modelled.

`JULES_SURFACE_TYPES::`**`ncpft`**

Type
integer

Permitted
< npft

Default
0

The number of crop plant functional types to be modelled.

`JULES_SURFACE_TYPES::`**`nnvg`**

Type
integer

Permitted
>= 1

Default
-32768

The number of non-plant surface types to be modelled.

Non-vegetated surface types

A negative value, when permitted, indicates that the surface type is not in use.

`JULES_SURFACE_TYPES::`**`urban`**

Type
integer

Permitted
-1, npft+1:ntype

Default
-32768

Index of the urban surface type (#6).

Can only be used if *l_urban2t* = FALSE.

JULES_SURFACE_TYPES::**lake**

Type
integer

Permitted
npft+1:ntype

Default
-32768

Index of the lake surface type (#7).

JULES_SURFACE_TYPES::**soil**

Type
integer

Permitted
npft+1:ntype

Default
-32768

Index of the soil surface type (#8).

Note: A soil surface type must be given (although the fraction may be set to zero).

JULES_SURFACE_TYPES::**ice**

Type
integer

Permitted
npft+1:ntype

Default
-32768

Index of the ice surface type (#9).

Note: In the UM the ice surface type must be specified (although the fraction may be set to zero).

Multiple ice tiles allowed to exist in an ice gridbox

These surface types can only be used when multiple ice tiles are allowed in a gridbox i.e. when *l_elev_land_ice* = TRUE.

JULES_SURFACE_TYPES::**elev_ice**

Type
integer

Permitted
-1,npft+1:ntype

Default
-32768

Indices of the elevated ice types (#901-925).

Must be grouped together with values `npft < elev_ice <= ntype` OR `elev_ice = -1` to indicate they are not used (i.e. all elevated rock instead).

`JULES_SURFACE_TYPES::`**`elev_rock`**

Type
integer

Permitted
-1,npft+1:ntype

Default
-32768

Indices of the elevated non-glaciated bedrock types (#926-950).

Must be grouped together, with values `npft < elev_rock <= ntype` OR `elev_rock = -1` to indicate they are not used (i.e. all elevated ice instead).

Two-tile urban schemes including MORUSES

These surface types can only be used when *`l_urban2t`* = TRUE.

`JULES_SURFACE_TYPES::`**`urban_canyon`**

Type
integer

Permitted
npft+1:ntype

Default
-32768

Index of the urban canyon surface type (#601).

`JULES_SURFACE_TYPES::`**`urban_roof`**

Type
integer

Permitted
npft+1:ntype

Default
-32768

Index of the urban roof surface type (#602).

Note: When giving urban fraction data (see *`JULES_FRAC`*), total *urban* fraction may be given instead of the separate canyon and roof fractions by entering it under the canyon fraction. When initialising if the roof fraction is zero, the canyon fraction will be interpreted as the total *urban* fraction and be partitioned according to the canyon fraction (W/R, see *`URBAN_PROPERTIES`*).

Surface types with `compulsory=false`

These are required to allow the surface type configuration to be checked at runtime and for surface types to be identified in the output headers. These are added as a latent variable. Remove the surface type if it is not required (see explanation at the *top* of this page).

JULES_SURFACE_TYPES::**usr_type**

Type
integer

Permitted
1:ntype

Default
-32768

Index of user specified surface type (#10-99).

A user surface type can be used when experimenting with new surface configurations without a code change. These can be either vegetated or non-vegetated and are used solely to assign an ID number.

Vegetated surface types

A negative value, when permitted, indicates that the surface type is not in use.

JULES_SURFACE_TYPES::**brd_leaf**

Type
integer

Permitted
1:npft

Default
-32768

Index of the original broadleaf PFT surface type (#1).

JULES_SURFACE_TYPES::**brd_leaf_dec**

Type
integer

Permitted
1:npft

Default
-32768

Index of broadleaf (decidous) PFT surface type (#101)

JULES_SURFACE_TYPES::**brd_leaf_eg_trop**

Type
integer

Permitted
1:npft

Default
-32768

Index of broadleaf (evergreen tropical) PFT surface type (#102).

JULES_SURFACE_TYPES::**brd_leaf_eg_temp**

Type
integer

Permitted
1:npft

Default
-32768

Index of broadleaf (evergreen temperate) PFT surface type (#103).

JULES_SURFACE_TYPES::**ndl_leaf**

Type
integer

Permitted
1:npft

Default
-32768

Index of original needleleaf PFT surface type (#2).

JULES_SURFACE_TYPES::**ndl_leaf_dec**

Type
integer

Permitted
1:npft

Default
-32768

Index of needleleaf (deciduous) PFT surface type (#201).

JULES_SURFACE_TYPES::**ndl_leaf_eg**

Type
integer

Permitted
1:npft

Default
-32768

Index of needleleaf (evergreen) PFT surface type (#202).

JULES_SURFACE_TYPES::**c3_grass**

Type
integer

Permitted
1:npft

Default
-32768

Index of original C3 grass PFT surface type (#3).

JULES_SURFACE_TYPES::**c3_crop**

Type
integer

Permitted
1:npft

Default
-32768

Index of C3 crop PFT surface type (#301).

JULES_SURFACE_TYPES::**c3_pasture**

Type
integer

Permitted
1:npft

Default
-32768

Index of C3 pasture PFT surface type (#302).

`JULES_SURFACE_TYPES::c4_grass`

Type
integer

Permitted
1:npft

Default
-32768

Index of original C4 grass PFT surface type (#4).

`JULES_SURFACE_TYPES::c4_crop`

Type
integer

Permitted
1:npft

Default
-32768

Index of C4 crop PFT surface type (#401).

`JULES_SURFACE_TYPES::c4_pasture`

Type
integer

Permitted
1:npft

Default
-32768

Index of C4 pasture PFT surface type (#402).

`JULES_SURFACE_TYPES::shrub`

Type
integer

Permitted
1:npft

Default
-32768

Index of original shrub PFT surface type (#5).

`JULES_SURFACE_TYPES::shrub_dec`

Type
integer

Permitted
1:npft

Default
-32768

Index of shrub (deciduous) PFT surface type (#501).

JULES_SURFACE_TYPES::**shrub_eg**

Type
integer

Permitted
1:npft

Default
-32768

Index of shrub (evergreen) PFT surface type (#502).

6.4 cable_surface_types.nml

This file configures the surface types used by CABLE. It contains one namelist called *CABLE_SURFACE_TYPES*.

6.4.1 CABLE_SURFACE_TYPES namelist members

CABLE_SURFACE_TYPES::**npft_cable**

Type
integer

Permitted
>= 1

Default
-32768

The number of plant functional types to be modelled.

CABLE_SURFACE_TYPES::**nnvg_cable**

Type
integer

Permitted
>= 1

Default
-32768

The number of non-plant surface types to be modelled.

Note: The total number of surface types to be modelled is called `ntype_cable`, and is given by `ntype_cable = npft_cable + nnvg_cable`.

In the standard setup, CABLE models 13 vegetation types and 4 non-vegetation types (`npft_cable = 13`, `nnvg_cable = 4`). However, the model domain need not contain all 13 types, e.g. the domain could consist of a single point with 100% grass. The amount of each type in the domain is normally set in *JULES_FRAC*.

CABLE_SURFACE_TYPES::**urban_drive**

Type
integer

Default
-32768

`CABLE_SURFACE_TYPES::`**`lakes_cable`**

Type
integer

Default
-32768

Index of the lakes surface type.

A negative value indicates no lakes surface type.

`CABLE_SURFACE_TYPES::`**`barren_cable`**

Type
integer

Permitted
>= 1

Default
-32768

Index of the barren soil surface type.

Note: A barren soil surface type must be given.

`CABLE_SURFACE_TYPES::`**`ice_cable`**

Type
integer

Default
-32768

Index of the ice surface type.

A negative value indicates no ice surface type.

6.5 `model_environment.nml`

This file sets the model environment options e.g. whether JULES is coupled to the UM or run in a standalone environment. It contains one namelist called *`JULES_MODEL_ENVIRONMENT`*.

There are many JULES science options that are in shared namelists, so they can be read both by standalone and by a model driving JULES e.g. the UM. However some options either make no scientific sense or the necessary input data are not available to the environment in which JULES is being driven as the plumbing has not yet been done. This causes problems for example when creating standalone apps from UM configurations. This namelist allows the environment in which JULES is being run to be specified so that options that are unavailable can be made inaccessible via the metadata and thus will not appear in the gui. Warnings can also be issued if options are inappropriately set.

This namelist also describes the flavour of the land surface model being used. CABLE is in the process of being incorporated into JULES and other flavours of JULES is in development e.g. a standalone rivers app.

6.5.1 `JULES_MODEL_ENVIRONMENT` namelist members

`JULES_MODEL_ENVIRONMENT::`**`l_jules_parent`**

Type
integer

Default
imdi

Switch to identify the environment in which JULES is being run. The switch should only be used to allow science options, which are not available in the specified model environment, to be trigger ignored and checked that they are set appropriately at run-time.

0	JULES is being run standalone. Any options that are only available to the parent model (e.g. the UM) will be trigger ignored.
1	JULES is being run coupled to the UM.
2	JULES is being run coupled via OASIS (available to Rivers-only executable only). Options not available to the UM are trigger-ignored.

Warning: No science code should be associated with this switch, only what science options are available.

Note: The metadata of the parent model only actually allows the appropriate option to be specified i.e. in standalone only 0 is permitted and in the UM only 1 is permitted. Any other parent models are listed here for information only. It is not appropriate to include a list of the unavailable options here. However, information for namelists that have been consolidated will appear in the following checking routines as they are completed.

- src/control/standalone/check_unavailable_options_mod.F90
- src/control/um/check_jules_unavailable_options_mod.F90

`JULES_MODEL_ENVIRONMENT::`**`lsm_id`**

Type
integer

Default
MDI

Switch for land surface model flavour.

1	JULES land surface model
2	CABLE land surface model

Note: The CABLE model has not yet been implemented within the JULES repository.

6.6 `jules_surface.nml`

This file sets the surface options. It contains one namelist called *JULES_SURFACE*.

6.6.1 `JULES_SURFACE` namelist members

`JULES_SURFACE::`**`all_tiles`**

Type
integer

Permitted
0,1

Default
0

Perform calculations of tile properties on all tiles (except land ice) for all gridpoints even when the tile fraction is zero.

0. Off
1. On

`JULES_SURFACE::`**`cor_mo_iter`**

Type
integer

Permitted
1-4

Default
1

Corrections to Monin-Obukhov surface exchange calculation. Please see also UMDP24 "The Parametrization of Boundary Layer Processes" (section 8.4.1).

1. Correct convective gustiness in low winds
2. Correct U* in dust scheme,
3. Limit Obukhov length in low winds
4. Improve the initialisation of the iteration

Note: Option 4 should be the preferred option.

`JULES_SURFACE::`**`beta_cnv_bl`**

Type
real

Permitted
>=0.0

Dimensionless coefficient scaling the boundary layer convective gustiness contribution to surface exchange. Historically this was set to 0.08 but is recommended to be reduced to 0.04 when gustiness from convective downdraughts is included, either from the convection parametrization or when convection is resolved (so resolutions ~1km or finer). Please see also UMDP24 "The Parametrization of Boundary Layer Processes" (section 8.1).

`JULES_SURFACE::`**`l_aggregate`**

Type
logical

Default
F

Switch controlling number of surface tiles for each gridbox.

This is used to set the number of surface energy balances that are solved for each gridbox (`nsurft`).

TRUE
Aggregate parameter values are used to solve a single energy balance per gridbox. This option sets `nsurft = 1`.

FALSE
A separate energy balance is calculated for each surface type. This option sets `nsurft = ntype`.

`JULES_SURFACE::i_aggregate_opt`

Type
integer

Permitted
0-1

Default
0

Option for aggregating surface properties to surface tiles:

0. Aggregate momentum roughness lengths and set the thermal roughness length as a given fraction of this (in practice the ratio of roughness lengths for the first surface type).
1. Aggregate the thermal roughness lengths separately from the momentum roughness lengths using an analogous algorithm.

Note: This option is ignored unless *`l_aggregate`* is true.

`JULES_SURFACE::l_epot_corr`

Type
logical

Default
F

TRUE
Use correction to the calculation of potential evaporation.

FALSE
No effect.

`JULES_SURFACE::l_point_data`

Type
logical

Default
F

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.

`JULES_SURFACE::l_land_ice_imp`

Type
logical

Default
F

Switch to control the use of implicit numerics to update land ice temperatures.

TRUE
Use implicit numerics to update land ice temperatures.

FALSE
Use explicit numerics to update land ice temperatures.

`JULES_SURFACE::l_anthrop_heat_src`

Type
logical

Default
F

Switch for inclusion of anthropogenic contribution to the surface heat flux from *urban* surface types. If `l_urban2t` then the anthropogenic heat will be distributed between the `urban_canyon` and `urban_roof` according to `anthrop_heat_scale`, otherwise it is added to `urban` only.

TRUE
Add anthropogenic effect.

FALSE
No effect.

`JULES_SURFACE::iscrntdiag`

Type
integer

Permitted
0-3 (standalone: 0 or 1 only)

Default
0

Switch controlling method for diagnosing screen temperature.

0. Use surface similarity theory (no decoupling).
1. Use surface similarity theory but allow decoupling in very stable conditions based on the quasi-equilibrium radiative solution.
2. Diagnose the screen temperature including transient effects and radiative cooling.
3. Diagnose the screen temperature and humidity including transient effects and radiative cooling. The diagnosis of the screen temperature follows option 2. This is an experimental option and is undergoing development and additional testing.

Note: Option 0 should be the preferred option in standalone i.e. no decoupling until the decoupled options are fully tested in standalone scenarios.

`JULES_SURFACE::l_elev_lw_down`

Type
logical

Default
false

If surface tiles are set to be at an elevation offset from the gridbox mean altitude (see *JULES_SURF_HGT*) this switch controls whether downwelling longwave radiation is adjusted along with surface air temperature and relative humidity.

If true, the downwelling longwave for each surface tile not at the gridbox mean height is adjusted by an amount proportional to the fourth power of the adjustment that has been made to the surface air temperature. The adjustments are then scaled such that the sum over all surface tiles conserves the gridbox mean energy in the original forcing.

`JULES_SURFACE::`**`l_elev_land_ice`**

Type
logical

Default
false

Allows multiple ice surface tiles to exist in an ice gridbox, usually with each representing a different elevation (*JULES_SURF_HGT*) band on in icesheet areas so that a sub-gridscale surface mass balance term (a strong function of altitude) can be derived for forcing icesheet/glacier models. When enabled, ice tiles in a gridbox do not use the usual (gridbox mean) JULES soil/ice subsurface model, but each tile has an independent single layer bedrock-type solid ice boundary condition under the snowpack.

In addition, when selected, dense snowpacks on elevated ice gridboxes are parameterised to behave more like firn in two ways: 1) The meltwater-holding capacity of snow layers reduces as a linear function of their density, becoming zero above the pore-closure density of 850 kg/m^2 so as to restrict retention of melt within the snowpack. 2) Where the top few centimetres of the pack has a density appropriate to firn/bare ice and the grain-size physics otherwise used for snow albedo become less appropriate, surface albedo becomes a function of density, tending towards that of bare ice as density increases (see *rho_firn_albedo*, *amax*, *aicemax*).

If this scheme is enabled, a depth for the bedrock layer must be provided (*dzsoil_elev*) and the new tile numbers must be specified (*JULES_SURFACE_TYPES*) as either type *elev_ice* (for fully glaciated areas) or *elev_rock* (for non-glaciated areas where the bedrock may become exposed under a thin snow layer). The total number of non-vegetated surface tiles, and their surface properties (*JULES_NVEGPARM*, usually set to be the same as the normal ice tile) must be set accordingly, as with any surface tile.

`JULES_SURFACE::`**`l_flake_model`**

Type
logical

Default
false

Switch for using the freshwater lake model 'FLake' on the lake/inland-water surface tile. More information on the FLake model can be found on the FLake website. A description of how FLake is coupled to JULES can be found in Rooney and Jones 2010.

When using FLake, it is not necessary to use a canopy representation of lake properties so *catch_nvg_io*, *ch_nvg_io* and *vf_nvg_io* should all be set to zero for the lake tile.

`JULES_SURFACE::`**`l_urban2t`**

Type
logical

Default
false

Switch for using the two-tile urban schemes (including MORUSES). This allows two urban surface tiles (*urban_canyon* and *urban_roof*) to be used instead of one. Additional parameters must be supplied via *JULES_NVEGPARM*, with some able to be provided by MORUSES (see *JULES_URBAN*).

`JULES_SURFACE::`**`l_mo_buoyancy_calc`**

Type
logical

Default
false

Default JULES (l_mo_buoyancy_flux = false) uses the buoyancy from the previous timestep to calculate the surface transfer coefficients. In coupled simulations this can lead to unrealistic surface temperatures if the stability suddenly switches from stable to unstable, due to the low turbulence determined by the stable buoyancy flux.

With the interactive buoyancy flux option (l_mo_buoyancy_flux = true) the surface energy balance and buoyancy flux are calculated within the iterative calculation for the Monin-Obukhov similarity theory for the surface exchange coefficients. On occations when the stability is around neutral it is possible that the iterative calculation does not converge. In this case the larger of the last two calculated transfer coefficients is then used to prevent any unrealistic surface temperatures.

Surface parameters

`JULES_SURFACE::`**`hleaf`**

Type
real

Default
5.7e4

Specific heat capacity of leaves (J K^{-1} per kg carbon).

See Hadley Centre Technical Note 30, p6, available from the Met Office Library.

`JULES_SURFACE::`**`hwood`**

Type
real

Default
1.1e4

Specific heat capacity of wood (J K^{-1} per kg carbon).

See Hadley Centre Technical Note 30, p6, available from the Met Office Library.

`JULES_SURFACE::`**`beta1`**

Type
real

Default
0.83

Coupling coefficient for co-limitation in photosynthesis model.

See Cox et al. (1999), Eq.61.

`JULES_SURFACE::`**`beta2`**

Type
real

Default
0.93

Coupling coefficient for co-limitation in photosynthesis model.

See Cox et al. (1999), Eq.61.

`JULES_SURFACE::`**`fwe_c3`**

Type
real

Default
0.5

Constant in expression for limitation of photosynthesis by transport of products, for C3 plants.

See Cox et al. (1999) Eq.60.

`JULES_SURFACE::`**`fwe_c4`**

Type
real

Default
20000.0

Constant in expression for limitation of photosynthesis by transport of products, for C4 plants.

See Cox et al. (1999) Eq.60.

6.7 `jules_radiation.nml`

This file sets the radiation options. It contains one namelist called *`JULES_RADIATION`*.

6.7.1 `JULES_RADIATION` namelist members

`JULES_RADIATION::`**`l_cosz`**

Type
logical

Default
T

Switch for calculation of solar zenith angle.

TRUE
Calculate zenith angle.

FALSE
Assume constant zenith angle of zero, meaning sun is directly overhead.

n.b. assuming that the sun is directly overhead may overestimate primary productivity if *`l_triffid`* = TRUE (see GPP on *JULES Output variables*).

`JULES_RADIATION::`**`l_spec_albedo`**

Type
logical

Default
F

Switch for the two-stream spectral land-surface albedo model.

TRUE
Use spectral albedo with VIS and NIR components.

FALSE
Use a single (averaged) waveband albedo.

`JULES_RADIATION::`**`l_spec_alb_bs`**

Type
logical

Default
F

Switch for albedo model, when spectral albedo is being used.

Requires *l_spec_albedo* = TRUE.

TRUE
Produces a single albedo for use by both the direct and diffuse beams (a 'blue' sky albedo). This currently copies the diffuse beam albedo for the direct beam.

FALSE
Produces both a direct ('black' sky) and a diffuse ('white' sky) albedo.

JULES_RADIATION::**l_niso_direct**

Type
logical

Default
F

Switch for using full non-isotropic expression for direct scattering in plant canopies when using the two-stream canopy radiation model.

Requires *l_spec_albedo* = TRUE.

TRUE
Use full non-isotropic expression for scattering in plant canopies.

FALSE
Use the original isotropic expression.

JULES_RADIATION::**l_snow_albedo**

Type
logical

Default
F

Switch for using prognostic snow properties, which represents the effect of snow aging and soot deposition, in model albedo.

Requires *l_spec_albedo* = TRUE.

TRUE
Use prognostic snow properties for albedo.

FALSE
Calculate albedo of snow using only snow depth.

JULES_RADIATION::**l_embedded_snow**

Type
logical

Default
F

Switch to account for pft LAI and pft height in calculation of snow albedo.

TRUE
Use the embedded canopy snow albedo model. This is exclusive of *l_snow_albedo*.

FALSE
No effect.

JULES_RADIATION::**l_mask_snow_orog**

Type
logical

Default
F

Switch for orographic masking of snow, which decreases the albedo of snow in mountainous regions.

TRUE
Include orographic masking of snow in calculating albedo.

FALSE
No effect.

`JULES_RADIATION::`**`l_albedo_obs`**

Type
logical

Default
F

Switch for applying a scaling factor to the albedo values, on surface tiles, so that the resultant aggregate albedo matches observations. The supplied albedos should be from an observed climatology or analysis system and be supplied via an ancillary file.

TRUE
Scale the albedo values on tiles within the physical limits supplied in *`JULES_PFTPARM`* and *`JULES_NVEGPARM`*. When *`l_spec_albedo`* = TRUE, VIS and NIR components are required and when *`l_spec_albedo`* = FALSE the single (averaged) waveband albedo is required.

Note: Observed albedo(s) must be prescribed in *prescribed_data.nml*.

FALSE
Do not scale the albedo values on tiles.

`JULES_RADIATION::`**`l_spec_sea_alb`**

Type
logical

Default
F

Switch to use spectrally varying open sea albedos

TRUE
When *`i_sea_alb_method`* = 1 or 2, spectrally varying sea albedos are produced only when the spectral file contains 6 SW bands identical to those used in HadGEM1.

When *`i_sea_alb_method`* = 3, the spectral variability is calculated as per the Jin et al. (2011) parameterisation.

FALSE
Uses the calculated broadband sea albedo instead.

`JULES_RADIATION::`**`i_sea_alb_method`**

Type
integer

Default
None

Choice of model for the Ocean Surface Albedo (open water, ice free)

1. Diffuse albedo constant (0.06), direct albedo from Briegleb and Ramanathan (1982).
2. Diffuse albedo constant (0.06), direct albedo from Barker and Li (1995).
3. Direct and diffuse albedo from Jin et al. (2011).
4. Fixed global value, defined by *`fixed_sea_albedo`*.
5. Fixed global value, defined by *`fixed_sea_albedo`*, above 271K and variable below this to simulate sea-ice following Liu et al. (2007), Joshi & Haberle (2012) and Turbet et al. (2016).

`JULES_RADIATION::`**`fixed_sea_albedo`**

Type
real

Default
None

The global value of sea albedo to use if *`i_sea_alb_method`* = 4, 5

`JULES_RADIATION::`**`wght_alb`**

Type
real(4)

Default
MDI

Weights to form the overall albedo from its components (VIS direct, VIS diffuse, NIR direct, NIR diffuse) (Ideally, if *`l_partition_albsoil`* = T, *`wght_alb`* and *`swdn_frac_albsoil`* should be consistent, with *`swdn_frac_albsoil`* equal to $\sum_{3,4}$ *`wght_alb`* / $\sum_1^4$ *`wght_alb`*. However, *`swdn_frac_albsoil`* is applied only to bare soil and having a single parameter is more transparent to the user, while *`wght_alb`* is used only in diagnostics in standalone JULES and may have historical settings. Hence, the consistency of these two variables is not enforced.)

`JULES_RADIATION::`**`l_hapke_soil`**

Type
logical

Default
F

Switch to enable Hapke's model of soil albedo to include a zenith-angle dependence

TRUE
Apply a zenith-angle dependence to the direct albedo.

FALSE
Use the diffuse albedo for the direct beam as well.

`JULES_RADIATION::`**`l_partition_albsoil`**

Type
logical

Default
F

Switch to apply a spectral partitioning to the soil albedo.

TRUE
Partition the soil albedo between the visible and near infrared parts of the spectrum using *`ratio_albsoil`* and *`swdn_frac_albsoil`*.

FALSE
Apply the broadband albedo in both spectral regions.

`JULES_RADIATION::`**`ratio_albsoil`**

Type
real

Default
MDI

Ratio of the NIR to the VIS albedo of bare soil. Used if *`l_partition_albsoil`* = T.

`JULES_RADIATION::`**`swdn_frac_albsoil`**

Type
real

Default
MDI

The fraction of the total downward SW radiation assumed to be in the NIR part of the spectrum for partitioning the soil albedo. Used if *`l_partition_albsoil`* = T. (Ideally, *`wght_alb`* and *`swdn_frac_albsoil`* should be consistent, with *`swdn_frac_albsoil`* equal to $\sum_{3,4}$ *`wght_alb`* / $\sum_{1}^{4}$ *`wght_alb`*. However, *`swdn_frac_albsoil`* is applied only to bare soil and having a single parameter is more transparent to the user, while *`wght_alb`* is used only in diagnostics in standalone JULES and may have historical settings. Hence, the consistency of these two variables is not enforced.)

See also:

References:

- Barker, H.W. and Li, Z. (1995), Improved Simulation of Clear-Sky Shortwave Radiative Transfer in the CCC-GCM. J. Climate, 8, 2213–2223, doi:10.1175/1520-0442(1995)008<2213:ISOCSS>2.0.CO;2
- Briegleb, B. and Ramanathan, V. (1982), Spectral and Diurnal Variations in Clear Sky Planetary Albedo. J. Appl. Meteor., 21, 1160–1171, doi:10.1175/1520-0450(1982)021<1160:SADVIC>2.0.CO;2
- Liu, J. , Zhang, Z. , Inoue, J. and Horton, R. M. (2007), Evaluation of snow/ice albedo parameterizations and their impacts on sea ice simulations. Int. J. Climatol., 27: 81-91. doi:10.1002/joc.1373
- Zhonghai Jin, Yanli Qiao, Yingjian Wang, Yonghua Fang, and Weining Yi, "A new parameterization of spectral and broadband ocean surface albedo", Opt. Express 19, 26429-26443 (2011), doi:10.1364/OE.19.026429
- B. Hapke, "Bidirectional reflectance spectroscopy: 1. Theory", J. Geophys. Res. 86(B4), 3039-3054 (1981), doi:10.1029/JB086iB04p03039
- Manoj M. Joshi and Robert M. Haberle. Astrobiology. Jan 2012. ahead of print doi:10.1089/ast.2011.0668
- Martin Turbet, Jérémy Leconte, Franck Selsis, Emeline Bolmont, François Forget, Ignasi Ribas, Sean N. Raymond and Guillem Anglada-Escudé (2016), The habitability of Proxima Centauri b - II. Possible climates and observability, A&A, 596, A112, doi:10.1051/0004-6361/201629577

6.8 `jules_hydrology.nml`

This file sets the hydrology options. It contains one namelist called *`JULES_HYDROLOGY`*.

6.8.1 `JULES_HYDROLOGY` namelist members

`JULES_HYDROLOGY::`**`l_top`**

Type
logical

Default
F

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.

See also:

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.

`JULES_HYDROLOGY::`**`l_pdm`**

Type
logical

Default
F

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.

See also:

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.

Note: Setting *`l_top`* = FALSE and *`l_pdm`* = FALSE selects a more basic runoff production scheme. In this scheme, surface runoff comes only from infiltration excess runoff (no saturation excess runoff), and subsurface runoff comes only from free drainage from the deepest soil layer (no lateral flow from mid-layers), as described in Essery et al. (2001, HCTN 30).

`JULES_HYDROLOGY::`**`l_limit_gsoil`**

Type
logical

Default
F

TRUE
Limit the soil conductance to the value when the top layer soil moisture is at the critical soil moisture. Below this threshold, the soil conductance follows Best et al. (2011) equation 7.

FALSE
Allow the soil conductance to increase as the top layer soil moisture goes above the critical soil moisture, as in Best et al. (2011) equation 7.

Only used if `l_top` = TRUE

JULES_HYDROLOGY::**zw_max**

Type
real

Default
None

The maximum allowed depth to the water table (m).

This is the depth from the soil surface to the bottom of an additional layer that is used to track water tables below the standard soil model (which has layer thicknesses given by *dzsoil_io*). A value of ~10m can often be used (though the previous default value was 6m) - the suitability of any value depends on values of the ancillary variable `fexp` (see *List of TOPMODEL parameters*) and the sum of the soil layer thicknesses (denoted *sum_dzsoil* here). The saturated hydraulic conductivity declines exponentially with depth in the additional deep TOPMODEL layer, with decay parameter `fexp`, and should be sufficiently small at depth `zw_max` that the flow at this depth can be neglected, that is *EXP(-fexp(zw_max-sum_dzsoil))* should be sufficiently small at all locations. (As a minimum guide, the code tests that the value of this expression is <= 0.05 and a warning is printed where this condition is not met; users should check model output logs for these messages.)

JULES_HYDROLOGY::**ti_max**

Type
real

Default
None

The maximum possible value of the topographic index. A value of 10.0 can be used.

JULES_HYDROLOGY::**ti_wetl**

Type
real

Default
None

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. A value of 1.5 can be used.

Note: When TOPMODEL is on (i.e. *l_top* = TRUE), JULES follows Gedney & Cox (2003, J Hydromet, eqn 14) in assuming that wetlands occur where gridcell elevation is low enough (assumed to be where topographic index is large enough) that the water table is above the land surface (topidx > *ti_wetl*) but not above the land surface by enough that streamflow may be assumed to occur (topidx < *ti_max*). Both *ti_wetl* and *ti_max* are levels calibrated from observed wetland fractions. So, if the water table is above the surface then JULES can calculate an areal fraction of total inundation (fsat) and also the areal fraction that is inundated but shallow enough to be stagnant/non-flowing (fwetl, with fwetl<=fsat), which is the 'wetland fraction'.

JULES_HYDROLOGY::**nfita**

Type
integer

Default
None

The number of values tried when fitting wetland and saturation fractions to water table depth in the initialisation. A value of 20 can be used.

This controls the range of `cfit` values tried in `calc_fit_fsat.F90` where `cfitmax = 0.15 * nfita`

`JULES_HYDROLOGY::`**`l_wetland_unfrozen`**

Type
logical

Default
F

TRUE
Treat the calculations of wetland and surface saturation fractions more like those of an unfrozen soil.

FALSE
Use standard wetland and surface saturation fraction calculations.

Only used if `l_pdm` = TRUE

`JULES_HYDROLOGY::`**`dz_pdm`**

Type
real

Default
None

The depth of soil considered by PDM (m).

A value of ~1m can be used.

`JULES_HYDROLOGY::`**`b_pdm`**

Type
real

Default
None

PDM shape parameter (exponent) of the Pareto distribution controlling spatial variability of storage capacity. A value ~1 can be used. b=0 implies a constant storage capacity at all points.

`JULES_HYDROLOGY::`**`l_spdmvar`**

Type
logical

Default
F

TRUE
Use a linear function of topographic slope to calculate S0/Smax (the minimum soil water storage below which there is no saturation excess runoff from PDM, expressed as a fraction of the maximum storage Smax): S0/Smax=MAX(0.0,1-(slope/*`slope_pdm_max`*)). The slope is read as an ancillary field (see *`JULES_PDM`*).

This function will result in high S0/Smax values for flatter regions and low values for steeper regions, and has been tested for catchments in Great Britain.

FALSE
Use a fixed value for S0/Smax, specified in *`s_pdm`*.

Only used if `l_spdmvar` = TRUE

`JULES_HYDROLOGY::`**`slope_pdm_max`**

Type
real

Default
None

The maximum topographic slope (deg) in the linear function of slope to calculate S0/Smax. Slopes above this value will result in a S0/Smax value of zero.

A value of 6.0 has been tested for slope fields calculated from a high resolution DEM dataset (50m IHDTM for Great Britain).

For slopes calculated from coarser DEM datasets, a lower value might be more appropriate as fine-resolution features of the terrain are not included.

Only used if `l_spdmvar` = FALSE

`JULES_HYDROLOGY::`**`s_pdm`**

type
real

permitted
0-1

default
None

Minimum soil water storage below which there is no saturation excess runoff from PDM, expressed as a fraction of the maximum storage Smax)

e.g. A value of 0 indicates that surface saturation can occur for any value of water storage. A value of 0.5 would indicate that no surface runoff is produced until the soil is 50% saturated.

6.9 `jules_soil.nml`

This file sets the soil options and parameters. It contains one namelist called *`JULES_SOIL`*.

6.9.1 `JULES_SOIL` namelist members

`JULES_SOIL::`**`sm_levels`**

Type
integer

Permitted
>= 1

Default
4

Number of soil layers.

A value of 4 is often used, with soil layer depths that have been tuned using this.

Warning: If *`ncpft`* > 0, `sm_levels >= 3` is required.

`JULES_SOIL::`**`l_vg_soil`**

Type
logical

Default
F

Switch for van Genuchten soil hydraulic model.

TRUE
Use van Genuchten model.

FALSE
Use Brooks and Corey model[1].

See also:

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.

`JULES_SOIL::l_dpsids_dsdz`

Type
logical

Default
F

Switch to calculate vertical gradient of soil suction with the assumption of linearity only for fractional saturation (consistent with the calculation of hydraulic conductivity).

`JULES_SOIL::l_soil_sat_down`

Type
logical

Default
F

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 (Down)
Any excess is put into the layer below. Any excess from the bottom layer becomes subsurface runoff.

FALSE (Up)
Any excess is put into the layer above. Any excess from the top layer becomes surface runoff. This option was used in JULES2.0.

`JULES_SOIL::l_holdwater`

Type
logical

Default
F

This switch fixes a problem in soil hydrology, whereby if a layer goes supersaturated during the implicit calulation, the excess water is pushed out of the soil column (`l_holdwater = FALSE`) instead of into an adjacent layer (`l_holdwater = TRUE`).

TRUE
Supersaturated soil moisture from implicit calculation goes into an adjacent layer (above or below depending on *l_soil_sat_down*). This option was added in JULES 5.1.

[1] 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 (Marthews et al. 2014,GMD). There has been confusion in the literature and in past documentation of MOSES/JULES, but JULES uses the Brooks and Corey model when *l_vg_soil* = FALSE .

References: * Brooks RH & Corey AT (1964). Hydraulic properties of porous media. Colorado State University Hydrology Papers 3. * Clapp RB & Hornberger GM (1978). Empirical Equations for Some Soil Hydraulic Properties. Water Resources Research 14:601-604.

FALSE
Supersaturated soil moisture from implicit calculation goes out of the base of the soil column.

`JULES_SOIL::soilhc_method`

Type
integer

Permitted
1, 2 or 3

Default
1

Switch for soil thermal conductivity model.

1. Use approach of Cox et al (1999), as in JULES2.0.

   This is likely to predict values of soil thermal conductivity that are too low (Dharssi et al, 2009).
2. Use approach of Dharssi et al (2009), which was adapted from Johansen (1975) and described by Peters-Lidard et al. (1998).

   This is not recommended for organic soils.
3. Use approach of Chadburn et al (2015).

   This is recommended when using organic soils, which can have a much lower saturated thermal conductivity than mineral soils.

See also:

References:

- Chadburn et al (2015). An improved representation of physical permafrost dynamics in a global land-surface scheme. Geoscientific Model Development
- Dharssi et al (2009). New soil physical properties implemented in the Unified Model at PS18. Met Office Technical note 528
- Johansen (1975). Thermal conductivity of soils. PhD thesis. University of Trondheim, Norway
- Peters-Lidard et al (1998). The effect of soil thermal conductivity parameterisation on surface energy fluxes and temperatures. J. Atmos. Sci. 55:1209-1224

`JULES_SOIL::l_bedrock`

Type
logical

Default
F

Switch for using a thermal bedrock column beneath the soil column. The bedrock has no hydrological processes - diffusion of heat is the only process represented.

Properties of the bedrock can be set using *ns_deep*, *hcapdeep*, *hcondeep* and *dzdeep*.

TRUE
An additional bedrock column is used below the soil column.

FALSE
No effect.

See also:

For full details see Chadburn et al. (2015)

Bedrock parameters (only used if `l_bedrock` = TRUE)

`JULES_SOIL::ns_deep`

Type
integer

Permitted
>= 1

Default
100

The number of levels in the thermal-only bedrock.

JULES_SOIL::**hcapdeep**

Type
real

Default
2100000.0

The heat capacity of the bedrock (J K^{-1} m^{-3}).

JULES_SOIL::**hcondeep**

Type
real

Default
8.6

The heat conductivity of the bedrock (W m^{-2} K^{-1}).

JULES_SOIL::**dzdeep**

Type
real

Default
0.5

The thickness of the bedrock layers (m).

JULES_SOIL::**cs_min**

Type
real

Default
1.0e-6

Minimum allowed soil carbon (kg m^{-2}).

JULES_SOIL::**zsmc**

Type
real

Permitted
> 0

Default
1.0

If a depth-averaged soil moisture diagnostic is requested, the average is calculated from the surface to this depth (m).

JULES_SOIL::**zst**

Type
real

Permitted
> 0

Default
1.0

The depth (0.0->zst) to which the soil temperature is averaged for use in the calculation of wetland methane emissions (m).

`JULES_SOIL::`**`confrac`**

Type
real

Permitted
0 <= confrac <= 1

Default
0.3

The fraction of the gridbox assumed to be covered by convective precipitation.

`JULES_SOIL::`**`dzsoil_io`**

Type
real(sm_levels)

Default
None

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.

It is recommended that JULES uses layer depths of 0.1, 0.25, 0.65 and 2.0m, giving a total depth of 3.0m, unless there is good reason not to.

`JULES_SOIL::`**`dzsoil_elev`**

Type
real

Default
None

Depth of the tiled solid-ice bedrock-type layer used underneath individual ice tiles if *`l_elev_land_ice`* is TRUE. Effectively this sets the amount of thermal buffering each surface tile has to heat fluxes penetrating through the snowpack.

`JULES_SOIL::`**`l_tile_soil`**

Type
logical

Default
False

Switch to set the number of soil tiles to equal the number of surface tiles. Each soil tile has independent properties.

See also *`l_broadcast_ancils`* and *`l_broadcast_soilt`*.

Note: Setting *`l_tile_soil`* = TRUE means a separate soil tile exists for each surface tile (rather than all surface tiles using the same, single soil tile). This also alters the names of many of the soil prognostic and ancillary variables that are used (see elsewhere), with the suffix "_soilt" being added to indicate the presence of soil tiling. The switches *`l_broadcast_ancils`* and *`l_broadcast_soilt`* allow soil tiling to be used with input files that do not contain soil tile information. Setting *`l_broadcast_ancils`* = TRUE means that a soil ancillary file that does not contain soil tiles can be used in a tiled run. Setting *`l_broadcast_soilt`* = TRUE means an initital state file that does not contain soil tiles can be used to initialise a run with soil tiles.

`JULES_SOIL::`**`l_broadcast_ancils`**

Type
logical

Default
False

Switch to allow non-soil tiled ancillary files to be broadcast to all soil tiles. Only active when *l_tile_soil* is True. When reading ancillaries from the dump file, use *l_broadcast_soilt* instead.

6.10 jules_vegetation.nml

This file sets the vegetation options. It contains one namelist called *JULES_VEGETATION*.

6.10.1 JULES_VEGETATION namelist members

JULES_VEGETATION::**l_trait_phys**

Type
logical

Default
F

Switch for using trait-based physiology.

TRUE
Vcmax is calculated based on observed leaf traits. Leaf nitrogen (nmass: kgN kgLeaf^{-1}) and leaf mass (LMA: kgLeaf m^{-2}) can be based on observations from the TRY database. Vcmax (umol CO_2 m^{-2} s^{-1}) is based on linear regressions as in *Kattge et al. 2009*. Two additional parameters are needed: vint and vsl - the intercept and slope, respectively, that relate the leaf nitrogen to vcmax. Sigl is replaced with LMA (sigl=LMA*Cmass, where Cmass is the kgC kgLeaf^{-1} and is 0.4).

FALSE
Vcmax is calculated based on parameters nl0 (kgN kgC^{-1}) and neff.

JULES_VEGETATION::**l_phenol**

Type
logical

Default
F

Switch for vegetation phenology model.

TRUE
Use phenology model.

FALSE
Do not use phenology model.

JULES_VEGETATION::**l_triffid**

Type
logical

Default
F

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 soil carbon pool is used.

JULES_VEGETATION::**l_veg_compete**

Type
logical

Default
T

Switch for competing vegetation.

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.

JULES_VEGETATION::**l_ht_compete**

Type
logical

Default
F

Only used if *l_triffid* = TRUE.

TRUE
Use height-based vegetation competition (recommended).

This allows for a generic number of PFTs. When *l_trif_eq* = TRUE, this is implemented by `lotka_eq_jls.F90`. When *l_trif_eq* = FALSE, it is implemented in `lotka_noeq_jls.F90` when *l_trif_crop* = FALSE and in `lotka_noeq_subset_jls.F90` when *l_trif_crop* = TRUE.

FALSE
Use the vegetation competition described in *HCTN24*.

This is hard-wired for 5 PFTs (BT, NT, C3, C4, SH, in that order) with co-competition for grasses and trees in `lokta_jls.F90`.

JULES_VEGETATION::**l_nitrogen**

Type
logical

Default
F

Only used if *l_triffid* = TRUE.

TRUE
Enable Nitrogen limitation of carbon uptake. A nitrogen deposition field should be provided otherwise no N deposition is assumed.

FALSE
No Nitrogen limitation. Nitrogen fluxes are calculated as diagnostics only.

JULES_VEGETATION::**l_trif_eq**

Type
logical

Default
T

Switch for equilibrium vegetation model (i.e., an equilibrium solution of TRIFFID).

Only used if *l_triffid* = TRUE.

TRUE
Use equilibrium TRIFFID.

FALSE
Do not use equilibrium TRIFFID.

`JULES_VEGETATION::`**`phenol_period`**

Type
integer

Permitted
>= 1

Default
None

Period for calls to phenology model in *days*. Only relevant if *`l_phenol`* = TRUE.

`JULES_VEGETATION::`**`triffid_period`**

Type
integer

Permitted
>= 1

Default
None

Period for calls to TRIFFID model in *days*. Only relevant if one of *`l_triffid`* or *`l_trif_eq`* is TRUE.

`JULES_VEGETATION::`**`l_gleaf_fix`**

Type
logical

Default
T

Switch for fixing a bug in the accumulation of `g_leaf_phen_acc`.

This bug occurs because `veg2` is called on TRIFFID timesteps and `veg1` is called on phenol timesteps, but `veg1` did not previously accumulate `g_leaf_phen_acc` in the same way as `veg2`.

TRUE
`veg1` accumulates `g_leaf_phen_acc` between calls to TRIFFID. This is important if *`triffid_period`* > *`phenol_period`*.

FALSE
`veg1` does not accumulate `g_leaf_phen_acc` between calls to TRIFFID.

`JULES_VEGETATION::`**`l_bvoc_emis`**

Type
logical

Default
F

Switch to enable calculation of BVOC emissions.

TRUE
BVOC emissions diagnostics will be calculated.

FALSE
BVOC emissions diagnostics will not be calculated.

`JULES_VEGETATION::`**`l_o3_damage`**

Type
logical

Default
F

Switch for ozone damage.

TRUE
Ozone damage is on.

Note: Ozone concentration in ppb must be prescribed in *prescribed_data.nml*.

FALSE
No effect.

`JULES_VEGETATION::l_stem_resp_fix`

Type
logical

Default
F

Switch for bug fix for stem respiration to use balanced LAI to derive respiring stem mass. The switch is included for backwards compatibility with existing configurations. Future updates should include this change.

TRUE
Respiring stem mass is derived allometrically.

FALSE
Respiring stem mass varies with seasonal LAI.

In the case of a Broadleaf tree in the winter (no leaves) this would mean stem respiration is scaled to 0.

`JULES_VEGETATION::l_scale_resp_pm`

Type
logical

Default
F

Scale whole plant maintenance respiration by the soil moisture stress factor, instead of only scaling leaf respiration.

TRUE
Soil moisture stress reduces leaf, root, and stem maintenance respiration.

FALSE
Soil moisture stress only reduces leaf maintenance respiration.

`JULES_VEGETATION::fsmc_shape`

Type
integer

Permitted
0,1

Default
0

Shape of soil moisture stress function on vegetation (fsmc).

0. Piece-wise linear in vol. soil moisture.
1. Piece-wise linear in soil potential. Currently only allowed when `const_z` = T and `l_use_pft_psi` = T.

Note: The option `fsmc_shape` = 1 is still in development. Users should ensure that results are as expected, and provide feedback where deficiencies are identified.

`JULES_VEGETATION::`**`l_use_pft_psi`**

Type
logical

Default
F

Switch for parameters in the soil moisture stress on vegetation function (fsmc).

TRUE
Fsmc is calculated from *`psi_close_io`* and *`psi_open_io`*.

FALSE
Fsmc is calculated from `sm_wilt` and `sm_crit` in *`JULES_SOIL_PROPS`* and *`fsmc_p0_io`*.

Note: Soil respiration and surface conductance of bare soil respectively will depend on `sm_wilt` and `sm_crit` in *`JULES_SOIL_PROPS`*, regardless of the setting of *`fsmc_shape`*.

Note: The option *`l_use_pft_psi`* = T is still in development. Users should ensure that results are as expected, and provide feedback where deficiencies are identified.

`JULES_VEGETATION::`**`l_vegcan_soilfx`**

Type
logical

Default
F

Switch for enhancement to canopy model to allow for conduction in the soil below the vegetative canopy, reducing coupling between the soil and the canopy.

TRUE
Allow for conduction in the soil.

FALSE
No effect.

`JULES_VEGETATION::`**`l_leaf_n_resp_fix`**

Type
logical

Default
F

Switch for bug fix for leaf nitrogen content used in the calculation of plant maintenance respiration. The switch is included for backwards compatibility with existing configurations. Runs with *`can_rad_mod`* = 1, 4 or 5 are affected.

TRUE
Use correct forms for canopy-average leaf N content.

FALSE
No effect.

`JULES_VEGETATION::`**`l_landuse`**

Type
logical

Default
F

Switch for using landuse change in conjunction with TRIFFID

Only used if *l_triffid* = TRUE.

TRUE
Land use change is implemented within TRIFFID. Litter fluxes are split between soil and wood product pools. Requires additional prognostics covering the product pools and the agricultural fraction from the previous TRIFFID call.

FALSE
All litter fluxes enter the soil

`JULES_VEGETATION::`**`l_recon`**

Type
logical

Default
T

Switch for reconfiguring vegetation fractions. Also initialises vegetation and soil biogeochemistry at land ice points. With the ECOSSE soil model this switch also ensures that the initial condition for soil biogeochemistry is internally consistent.

TRUE
For soil points (land points with no ice) ensure vegetation fractions are at least a minimum value and reduce other fractions accordingly.

FALSE
Do not apply the minimum vegetation fractions. This is useful when some points are 100% lake and urban, in which case reconfiguration leads to a total surface tile fraction of greater than 1.

`JULES_VEGETATION::`**`l_prescsow`**

Type
logical

Default
F

Switch that determines how crop sowing dates are defined. Only used if *ncpft* > 0.

TRUE
Sowing dates prescribed in *JULES_CROP_PROPS* are used.

FALSE
Sowing dates are determined by the model.

`JULES_VEGETATION::`**`l_trif_crop`**

Type
logical

Default
F

Switch controlling the treatment of agricultural PFTs. Where agricultural PFTs are defined by the *crop_io* parameter.

TRUE
In the non-agricultural area natural PFT competition is calculated by a call to a new version of the lotka routine and in each agricultural area agricultural-PFT competition is calculated by an additional call to the new version of the lotka routine. Crop and pasture areas are defined by the *frac_agr* and *frac_past* variables respectively. Additionally, to represent harvesting, a fraction of crop litter is added to the fast wood products pool instead of the soil carbon pools.

FALSE
Vegetation competition is calculated for natural and crop PFTs together, with natural PFTs excluded from the agricultural area that is defined by the *frac_agr* variable. Agricultural PFTs can also grow in natural areas where they are interpreted as natural grasses.

JULES_VEGETATION::**l_trif_biocrop**

type
: logical

default
: F

Allows for representation of bioenergy crops with continuous or periodic harvesting of agricultural PFTs at prescribed intervals. Requires *l_trif_crop* = TRUE.

TRUE
: Crop, pasture, and bioenergy crop areas are defined by the *frac_agr*, *frac_past*, *frac_biocrop* variables respectively. Harvests are permitted from any land class and enabled for each PFT separately using the *harvest_type_io* variable. Harvesting may be continuous (as per the existing scheme in *l_trif_crop*, when *harvest_type_io* is 1), or performed at prescribed intervals defined using the *harvest_freq_io* and *harvest_ht_io* variables (when *harvest_type_io* is 2).

FALSE
: Land use classes, PFT partitioning, and harvests are as defined by the *l_trif_crop* switch.

See also:

References:

- Littleton et al., 2020, JULES-BE: representation of bioenergy crops and harvesting in the Joint UK Land Environment Simulator vn5.1, Geosci. Model Dev., https://doi.org/10.5194/gmd-13-1123-2020

JULES_VEGETATION::**l_ag_expand**

Type
: logical

Default
: F

Allows for assisted expansion of agricultural crop areas. Requires *l_landuse* = TRUE.

TRUE
: Automatically plant out new crop areas with target PFTs.

FALSE
: No automatic increase of PFT fraction when land class fraction increases.

JULES_VEGETATION::**can_model**

Type
: integer

Permitted
: 1-4

Default
: 4

Choice of canopy model for vegetation:

1. No distinct canopy (i.e. surface is represented as a single entity for radiative processes).
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.

Note:

***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.

JULES_VEGETATION::**can_rad_mod**

Type
integer

Permitted
1, 4, 5, 6

Default
4

Options 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.

4. Multi-layer approach for radiation interception following the two-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. It has an exponential decline of leaf N through the canopy and includes inhibition of leaf respiration in the light. Canopy photosynthesis and conductance are calculated as the sum over all layers.
5. This is an improvement of option 4, including:
   - Sunfleck penetration though the canopy.
   - Division of sunlit and shaded leaves within each canopy level.
   - A modified version of inhibition of leaf respiration in the light.
6. This is an improvement of option 5, including an exponential decline of leaf N with canopy height proportional to LAI, following Beer's law.

Note: `can_rad_mod` = 1 and 6 are recommended.

Note: When using `can_rad_mod` = 4, 5 or 6 it is recommended to use driving data that contains direct and diffuse radiation separately rather than a constant diffuse fraction.

See also:

Descriptions of option 1 can be found in *Jogireddy et al. (2006)*, and an application of option 4 can be found in *Mercado et al. (2007)*. Options 1 to 5 are described in *Clark et al (2011)*.

JULES_VEGETATION::**ilayers**

Type
integer

Permitted
>= 0

Default
10

Number of layers for canopy radiation model. Only used for `can_rad_mod` = 4, 5 or 6.

These layers are used for the calculations of radiation interception and photosynthesis.

JULES_VEGETATION::**photo_model**

Type
integer

Permitted
1 or 2

Default
none

Choice for model of leaf photosynthesis.

Possible values are:

1. C_3 and C_4 plants use the models of Collatz et al., 1991 and 1992, respectively. These were used in the original JULES model.
2. C_3 plants use the model of Farquhar et al. (1980); C_4 plants use the model of Collatz et al. (1992).

> **Warning:** The Farquhar model can only be used if `can_rad_mod` = 1, 5 or 6. Code has not been written for other values of `can_rad_mod`.

See also:

References:

- Collatz et al., 1991, Physiological and environmental regulation of stomatal conductance, photosynthesis, and transpiration – a model that includes a laminar boundary layer, Agricultural and Forest Meteorology, https://doi.org/10.1016/0168-1923(91)90002-8.
- Collatz et al., 1992, Coupled Photosynthesis-Stomatal Conductance Model for Leaves of C_4 Plants, Australian Journal of Plant Physiology, https://doi.org/10.1071/PP9920519.
- Farquhar et al., 1980, A biochemical model of photosynthetic CO_2 assimilation in leaves of C_3 species, Planta, https://doi.org/10.1007/BF0038623.

`JULES_VEGETATION::`**`stomata_model`**

Type
integer

Permitted
1 or 2

Default
1

Choice for model of stomatal conductance.

Possible values are:

1. The original JULES model, including the Jacobs closure - see Eqn.9 of *Best et al. (2011)*.
2. The model of *Medlyn et al. (2011)* - see Eqn.11 of that paper, and *Medlyn et al (2012)*. Note that as implemented the model uses a single parameter (g_1, assuming that $g_0 = 0$).

> **Warning:** Only the original (Jacobs) model can currently be used with the UM (Option 1).

`JULES_VEGETATION::`**`frac_min`**

Type
real

Default
1.0e-6

Minimum fraction that a PFT is allowed to cover if TRIFFID is used.

`JULES_VEGETATION::`**`frac_seed`**

Type
real

Default
0.01

Seed fraction for TRIFFID.

`JULES_VEGETATION::`**`pow`**

Type
real

Default
5.241e-4

Power in sigmodial function used to get competition coefficients.

See Hadley Centre Technical Note 24, Eq.3.

`JULES_VEGETATION::`**`l_inferno`**

Type
logical

Default
F

Switch that determines whether interactive fires (INFERNO) is used. This allows for the diagnostic of burnt area, burnt carbon and a variety of fire emissions.

TRUE
INFERNO is used to provide diagnostic fire variables

FALSE
INFERNO is not used.

`JULES_VEGETATION::`**`ignition_method`**

Type
integer

Permitted
1, 2, 3

Default
1

Switch to determine the type of ignition used (ubiquitous or prescribed with population and lightning)

1. INFERNO uses ubiquitous (constant) ignitions, of 1.67 fires km^{-2} s^{-1} (1.5 from humans, 0.17 from lightning).
2. INFERNO uses prescribed lightning ignitions, either from an ancillary or the UM. Meanwhile humans are assumed to ignite 1.5 fires km^{-2} s^{-1}.
3. INFERNO uses prescribed ignition using Population Density and Lightning Frequency (Cloud-to-Ground). These must be provided as prescribed data to the JULES run.

`JULES_VEGETATION::`**`l_trif_fire`**

Type
logical

Default
F

Switch that determines whether interactive fire is used. This allows for burnt area to link with dynamic vegetation.

Only used if *`l_triffid`* = TRUE.

TRUE
Burnt area is calculated in INFERNO and passed to TRIFFID to calculate vegetation dynamics. Carbon is also removed from DPM and RPM pools in SOILCARB.

FALSE
Burnt area is zero unless prescribed via an ancillary file.

`JULES_VEGETATION::l_vegdrag_pft`

Type
logical(npft)

Default
F

Switch for using vegetation canopy drag scheme on each PFT.

TRUE
Use a vegetative drag scheme. This is based on *Harman and Finnigan (2007)*.

FALSE
Do not use vegetative drag scheme.

`JULES_VEGETATION::l_rsl_scalar`

Type
logical

Default
F

Switch for using a roughness sublayer correction scheme in scalar variables. This is based on *Harman and Finnigan (2008)*.

Only use if any `l_vegdrag_pft` = TRUE.

TRUE
Use a roughness sublayer correction scheme in scalar variables.

FALSE
Do not use a roughness sublayer correction scheme in scalar variables.

`JULES_VEGETATION::c1_usuh`

Type
real

Permitted
>= 0

Default
None

u*/U(h) at the top of dense canopy. See *Massman (1997)*.

Only use if any `l_vegdrag_pft` = TRUE.

`JULES_VEGETATION::c2_usuh`

Type
real

Permitted
>= 0

Default
None

u*/U(h) at substrate under canopy. See *Massman (1997)*.

Only use if any `l_vegdrag_pft` = TRUE.

`JULES_VEGETATION::c3_usuh`

Type
real

Permitted
>= 0

Default
None

This is used in the exponent of equation weighting dense and sparse vegetation to get u*/U(h) in neutral condition. See *Massman (1997)*. The default value is taken from *Wang (2012)*.

Only use if any `l_vegdrag_pft` = TRUE.

`JULES_VEGETATION::cd_leaf`

Type
real

Permitted
0:1

Default
None

Leaf level drag coefficient.

Only use if any `l_vegdrag_pft` = TRUE.

`JULES_VEGETATION::stanton_leaf`

Type
real

Permitted
0:1

Default
None

Leaf-level Stanton number

Only use if `l_rsl_scalar` = TRUE.

`JULES_VEGETATION::l_spec_veg_z0`

Type
logical

Default
F

Switch for using specified values of the vegetation roughness length rather than being determined by the canopy height.

TRUE
Vegetation roughness lengths are specified for each PFT in `z0v_io`.

FALSE
Vegetation roughness lengths are calculated using canopy heights and parameter `dz0v_dh_io`.

`JULES_VEGETATION::l_limit_canhc`

Type
logical

Default
F

Switch for limiting the canopy heat capacity for vegetation, which is calculated from the canopy height.

Using the SIMARD canopy height ancillary gives very large heat capacities in the Amazon, so this switch limits the areal heat capacity to 1.15e5 J kg^{-1} m^{-2}, which is the value calculated by the default broadleaf tree height of 19.01 m.

TRUE
Vegetation areal heat capacity limited.

FALSE
Vegetation areal heat capacity unlimited.

Only used with the Farquhar model of leaf photosynthesis (`photo_model` = 2).

`JULES_VEGETATION::`**`photo_acclim_model`**

Type
integer

Permitted
0, 1, 2, or 3

Default
None

Choice for model of thermal response of photosynthetic capacity. Possible values are:

0. No adaptation or acclimation.
1. Thermal adaptation - plant response to temperature varies geographically in response to a static "home" temperature.
2. Thermal acclimation - plant response to temperature varies geographically and temporally in response to a dynamic "growth" temperature.
3. Thermal adaptation and acclimation - plant response to temperature varies geographically and temporally in response to a static "home" temperature and a dynamic "growth" temperature.

Note: When *`photo_acclim_model`* = 1 or 3 is used, the user must supply the long-term home temperature as ancillary field `t_home_gb` in *`JULES_VEGETATION_PROPS`*. When *`photo_acclim_model`* = 2 or 3 is used, the user must supply the running mean growth temperature as initial condition `t_growth_gb` in *`JULES_INITIAL`*.

`JULES_VEGETATION::`**`photo_act_model`**

Type
integer

Permitted
1 or 2

Default
None

Choice of model for the activation energies of J_{max} and V_{cmax}.

1. Activation energies vary by PFT but not by land point, and are NOT subject to acclimation.
2. Activation energies vary by land point but not by PFT, and are subject to acclimation.

Note: When *`photo_act_model`* = 1 is used, activation energies are calculated using *`act_jmax_io`* and *`act_vcmax_io`*. When *`photo_act_model`* = 2 is used, activation energies are calculated using *`act_j_coef`* and *`act_v_coef`*.

Warning: A value of 1 (PFT-dependent) must be used if *`photo_acclim_model`* = 0 (no adaptation or acclimation).

`JULES_VEGETATION::`**`photo_jv_model`**

Type
integer

Permitted
1 or 2

Default
None

Choice for model of for the variation of J_{25}/V_{25}.

1. J_{25} is found by scaling V_{25} by the given ratio J_{25}/V_{25}, that is, all the variation in the ratio comes from varying J_{25} (while V_{25} remains fixed).
2. J25 and V25 are calculated assuming that the total amount of nitrogen allocated to photosynthesis remains constant, thus any change in J25 requires a compensatory change in V25 - as used in *Mercado et al. (2018)*.

Warning: A value of 1 (simple scaling) must be used if *`photo_acclim_model`* = 0 (no adaptation or acclimation).

Only used with `photo_jv_model` = 2.

`JULES_VEGETATION::`**`n_alloc_jmax`**

Type
real

Default
None

Constant relating nitrogen allocation to J_{max} (mol CO^2 m^{-2} s^{-1} [kg m^{-2}]$^{-1}$). This is 5.3 in Eq.5 of *Mercado et al. (2018)*.

`JULES_VEGETATION::`**`n_alloc_vcmax`**

Type
real

Default
None

Constant relating nitrogen allocation to V_{cmax} (mol CO^2 m^{-2} s^{-1} [kg m^{-2}]$^{-1}$). This is 3.8 in Eq.5 of *Mercado et al. (2018)*.

Only used with thermal adaptation or acclimation of photosynthesis (`photo_acclim_model` = 1, 2 or 3).

The thermal adaptation/acclimation scheme in JULES is structured following Eq. 13 of *Kumarathunge et al. (2019)*, in which C3 photosynthetic capacity is allowed to vary at each land point as a function of a static home temperature (T_h) and a dynamic growth temperature (T_g). This is achieved by calculating five parameters used in the Farquhar photosynthesis scheme as functions of those temperature fields, rather than using fixed parameters from *`JULES_PFTPARM`*. Each parameter, Q, is calculated as a linear function of T_h and T_g:

$Q(T_h, T_g) = Q_{coef}(0) + Q_{coef}(1)\ T_h + Q_{coef}(2)\ T_g$.

The following namelist members specify the coefficients, Q_{coef}, used for each parameter. Note that, in each case, the units for $Q_{coef}(1)$ and $Q_{coef}(2)$ have an extra factor K^{-1} relative to the units for $Q_{coef}(0)$. This structure can be configured to represent the acclimation scheme of *Kattge and Knorr (2007)*, as used by *Mercado et al. (2018)*, and the scheme of *Kumarathunge et al. (2019)*.

Note: If *photo_acclim_model* = 1 is used all Q_{coef}(2) must equal 0.0, and if *photo_acclim_model* = 2 is used all Q_{coef}(1) must equal 0.0.

JULES_VEGETATION::**act_j_coef**

Type
real(3)

Default
None

Coefficients for the activation energy for J_{max} (J mol^{-1} and J mol^{-1} K^{-1}). Replaces the use of *act_jmax_io*.

JULES_VEGETATION::**act_v_coef**

Type
real(3)

Default
None

Coefficients for the activation energy for V_{cmax} (J mol^{-1} and J mol^{-1} K^{-1}). Replaces the use of *act_vcmax_io*.

JULES_VEGETATION::**dsj_coef**

Type
real(3)

Default
None

Coefficients for entropy factor for J_{max} (J mol^{-1} K^{-1} and J mol^{-1} K^{-2}). Replaces the use of *deact_jmax_io*.

JULES_VEGETATION::**dsv_coef**

Type
real(3)

Default
None

Coefficients for the entropy factor for V_{cmax} (J mol^{-1} K^{-1} and J mol^{-1} K^{-2}). Replaces the use of *deact_vcmax_io*.

JULES_VEGETATION::**jv25_coef**

Type
real(3)

Default
None

Coefficients for the ratio J_{25}/V_{25} (mol electrons [mol^{-1} CO_2] and (mol electrons [mol^{-1} CO_2] K^{-1}). Replaces the use of *jv25_ratio_io*.

Only used with thermal acclimation of photosynthesis (photo_acclim_model = 2 or 3).

JULES_VEGETATION::**n_day_photo_acclim**

Type
real

Default
None

Time constant (days) for the exponential moving average of temperature that is used as the growth temperature. Given a step function as input, the smoothed output has fallen to 1/e (approx. 37%) of the initial value after this number of days.

`JULES_VEGETATION::l_croprotate`

Type
logical

Default
F

Switch that enables sequential cropping (crop rotations). Only used if *`ncpft`* > 0 and if *`l_prescsow`* = T.

TRUE
Sowing dates and latest harvest dates prescribed in *`JULES_CROP_PROPS`* are used. The method is implemented in *Mathison et al. (2019)*.

FALSE
The crop model is used in its standard form with a single crop per year

6.10.2 `JULES_VEGETATION` references

- Best et al., 2011, The Joint UK Land Environment Simulator (JULES), model description – Part 1: Energy and water fluxes, Geosci. Model Dev., https://doi.org/10.5194/gmd-4-677-2011.
- Clark et al., 2011, The Joint UK Land Environment Simulator (JULES) model description – Part 2: Carbon fluxes and vegetation dynamics, Geosci. Model Dev., 4, 701-722, https://doi.org/10.5194/gmd-4-701-2011
- Harman, I.N. & Finnigan, J.J. (2007), A simple unified theory for flow in the canopy and roughness sublayer. Boundary-Layer Meteorol. 123: 339. https://doi.org/10.1007/s10546-006-9145-6
- Harman, I.N. & Finnigan, J.J. (2008), Scalar Concentration Profiles in the Canopy and Roughness Sublayer. Boundary-Layer Meteorol. 129: 323. https://doi.org/10.1007/s10546-008-9328-4
- HCTN24, Hadley Centre Technical Note 24, available from the Met Office Library. For ease the direct link to this document is: HCTN24 "Description of the "TRIFFID" Dynamic Global Vegetation Model".
- Jogireddy, V., Cox, P. M., Huntingford, C., Harding, R. J., and Mercado, L. M.: An improved description of canopy light interception for use in a GCM land-surface scheme: calibration and testing against carbon fluxes at a coniferous forest, Hadley Centre Technical Note 63, Hadley Centre, Met Office, Exeter, UK, 2006. https://digital.nmla.metoffice.gov.uk/IO_7873ea05-61ec-4615-b030-6bc33397d675
- Kattge, J. and Knorr, W., 2007, Temperature acclimation in a biochemical model of photosynthesis: a re-analysis of data from 36 species, Plant, Cell and Environment, 30: 1176–1190, https://doi.org/10.1111/j.1365-3040.2007.01690.x.
- Kattge, J. , Knorr, W. , Raddatz, T. and Wirth, C. (2009), Quantifying photosynthetic capacity and its relationship to leaf nitrogen content for global-scale terrestrial biosphere models. Global Change Biology, 15: 976-991. https://doi.org/doi:10.1111/j.1365-2486.2008.01744.x
- Kumarathunge, D. P. et al (2019), Acclimation and adaptation components of the temperature dependence of plant photosynthesis at the global scale, New Phytologist, 222: 768-784, https://doi.org/10.1111/nph.15668
- Massman, W. J. (1997), An Analytical One-Dimensional Model of Momentum Transfer by Vegetation of Arbitrary Structure, Boundary-Layer Meteorol. 83: 407-421.
- Medlyn, B. E., Duursma, R. A., Eamus, D. , Ellsworth, D. S., Prentice, I. C., Barton, C. V., Crous, K. Y., De angelis, P., Freeman, M. and Wingate, L. (2011), Reconciling the optimal and empirical approaches to modelling stomatal conductance. Global Change Biology, 17: 2134-2144. https://doi.org/10.1111/j.1365-2486.2010.02375.x
- Medlyn, B. E., Duursma, R. A., Eamus, D. , Ellsworth, D. S., Prentice, I. C., Barton, C. V., Crous, K. Y., De angelis, P., Freeman, M. and Wingate, L. (2012), Reconciling the optimal and empirical approaches to modelling stomatal conductance. Global Change Biology, 18: 3476-3476. https://doi.org/10.1111/j.1365-2486.2012.02790.x.

- Mercado, L. M., Huntingford, C., Gash, J. H. C., Cox, P. M., and Jogireddy, V.: Improving the representation of radiative interception and photosynthesis for climate model applications, Tellus B, 59, 553–565, 2007. https://doi.org/10.1111/j.1600-0889.2007.00256.x
- Mercado et al., 2018, Large sensitivity in land carbon storage due to geographical and temporal variation in the thermal response of photosynthetic capacity, New Phytologist, 218: 1462–1477, https://doi.org/10.1111/nph.15100.
- Sellers et al., 1992, Canopy reflectance, photosynthesis, and transpiration. III. A reanalysis using improved leaf models and a new canopy integration scheme. Remote Sens. Environ., 42, 187-216, https://doi.org/10.1016/0034-4257(92)90102-P
- Wang, W. (2012), An Analytical Model for Mean Wind Profiles in Sparse Canopies. Boundary-Layer Meteorol 142: 383. https://doi.org/10.1007/s10546-011-9687-0
- Mathison, C , Challinor, A. J., Deva, C., Falloon, P., Garrigues, S., Moulin, S., Williams, K., and Wiltshire, A. (2019), Developing a sequential cropping capability in the JULESvn5.2 land–surface model, Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2019-85, in review, 2019.

6.11 `jules_soil_biogeochem.nml`

This file sets options and parameters for soil biogeochemistry.

If using the single-pool or 4-pool soil models, all soil parameters are read from this file.

If using the ECOSSE soil model, most soil parameters are read from a separate file (*jules_soil_ecosse.nml*).

6.11.1 `JULES_SOIL_BIOGEOCHEM` namelist members

`JULES_SOIL_BIOGEOCHEM::`**`soil_bgc_model`**

Type
integer

Permitted
1, 2 or 3

Default
1

Choice for model of soil biogeochemistry.

Possible values are:

1. A single-pool model of soil carbon turnover in which the pool is not prognostic (not updated).
   This must be used when the TRIFFID vegetation model is not selected (*`l_triffid`* = FALSE).
2. A 4-pool model of soil organic carbon and nitrogen, originally based on the Jenkinson (1990) model, with a single pool of inorganic N.
   Historically this was bundled with the TRIFFID vegetation model.
   This can only be used if the TRIFFID vegetation model is selected (*`l_triffid`* = TRUE).
3. A 4-pool model of soil organic carbon and nitrogen, and 2 inorganic N pools (ammonium and nitrate), based on the ECOSSE model (Smith et al., 2010).
   This can only be used if the TRIFFID vegetation model is selected (*`l_triffid`* = TRUE).
   This can also be run without nitrogen (*`l_soil_n`* = FALSE).

Warning: The ECOSSE model in JULES is still in development and is not fully functional in this version. The code is included to allow further development. Users should not try to use ECOSSE.

See also:

References:

- Jenkinson, D.S., 1990. The turnover of organic carbon and nitrogen in soil. Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences, 329(1255), pp.361-368. (https://doi.org/10.1098/rstb.1990.0177)
- Smith et al., 2010, Estimating changes in Scottish soil carbon stocks using ECOSSE. I. Model description and uncertainties, Climate Research, 45: 179-192. (https://doi.org/10.3354/cr00899).

Parameters that can be used with all soil models

`JULES_SOIL_BIOGEOCHEM::`**`q10_soil`**

Type
real

Default
2.0

Q10 factor for soil respiration.

With the single-pool or 4-pool models this is only used if *`l_q10`* = TRUE.

With the ECOSSE model this is only used if *`temp_modifier`* = 1.

See Hadley Centre Technical Note 24, Eq.17, available from the Met Office Library.

Parameters for the single-pool model (only used if `soil_bgc_model` = 1)

`JULES_SOIL_BIOGEOCHEM::`**`kaps`**

Type
real

Default
0.5e-8

Specific soil respiration rate at 25 degC and optimum soil moisture (s^{-1}).

See Hadley Centre Technical Note 24, Eq.16, available from the Met Office Library.

Parameters for the single-pool and 4-pool models (only used if `soil_bgc_model` = 1 or 2)

`JULES_SOIL_BIOGEOCHEM::`**`l_q10`**

Type
logical

Default
T

Switch for use of Q10 approach when calculating soil respiration.

TRUE

Use Q10 approach (Equation 65 in Clark et al., 2011).

Note: This is always enforced if the single-pool model is selected (*`soil_bgc_model`* = 1) and was used in JULES2.0.

FALSE
Use the approach of Jenkinson (1990) (Equation 66 in Clark et al., 2011).

See also:

References:

- Jenkinson, D.S., 1990. The turnover of organic carbon and nitrogen in soil. Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences, 329(1255), pp.361-368. (https://doi.org/10.1098/rstb.1990.0177)
- Clark, D. B., Mercado, L. M., Sitch, S., Jones, C. D., Gedney, N., Best, M. J., Pryor, M., Rooney, G. G., Essery, R. L. H., Blyth, E., Boucher, O., Harding, R. J., Huntingford, C., and Cox, P. M.: The Joint UK Land Environment Simulator (JULES), model description – Part 2: Carbon fluxes and vegetation dynamics, Geosci. Model Dev., 4, 701–722, (https://doi.org/10.5194/gmd-4-701-2011), 2011.

`JULES_SOIL_BIOGEOCHEM::`**`l_soil_resp_lev2`**

Type
logical

Default
F

Switch affecting the temperature and moisture used for soil respiration calculation.

TRUE
Temperature and total (frozen+unfrozen) moisture content of the second soil layer are used.

FALSE
Temperature and unfrozen moisture content of the first (topmost) soil layer are used.

Note: If layered soil C is used (*`l_layeredc`* = TRUE) the temperature and moisture of each soil layer is used to calculation respiration from that layer.

Parameters for the 4-pool model (only used if `soil_bgc_model` = 2)

`JULES_SOIL_BIOGEOCHEM::`**`l_layeredc`**

Type
logical

Default
F

Switch for using the layered soil carbon model.

If the 4-pool model is used (*`soil_bgc_model`* = 2) this uses the approach of Burke et al. (2017) and two extra parameters are required: *`tau_resp`*, *`tau_lit`*.

Layered soil nitrogen is also available if the nitrogen cycle is switched on (*`l_nitrogen`* = TRUE), but this is a highly experimental version which needs further evaluation and so should be used with extreme caution. One additional parameter is required for layered soil nitrogen: *`diff_n_pft`*.

TRUE
The number and thickness of layers in the soil carbon model are set equal to those in the soil moisture model (*`JULES_SOIL`*).

FALSE
There are no specific layers in the soil carbon model (a single, bulk pool).

See also:

References:

- Burke, E. J., Chadburn, S. E., and Ekici, A.: A vertical representation of soil carbon in the JULES land surface scheme (vn4.3_permafrost) with a focus on permafrost regions, Geosci. Model Dev., 10, 959-975, doi:10.5194/gmd-10-959-2017, 2017.

`JULES_SOIL_BIOGEOCHEM::`**`l_label_frac_cs`**

Type
logical

Default
F

Switch for labelling and tracing a subset of the layered soil carbon (*`l_layeredc`* = TRUE). It uses the approach of Burke et al.(2017). This requires the 4-pool model to be used (*`soil_bgc_model`* = 2). The fraction of labelled soil carbon needs to be specified as part of the model's initial state.

TRUE
A user-defined fraction of soil carbon is labelled.

FALSE
None of the soil carbon is labelled.

See also:

References:

- Burke, E. J., Chadburn, S. E., and Ekici, A.: A vertical representation of soil carbon in the JULES land surface scheme (vn4.3_permafrost) with a focus on permafrost regions, Geosci. Model Dev., 10, 959-975, doi:10.5194/gmd-10-959-2017, 2017.

`JULES_SOIL_BIOGEOCHEM::`**`kaps_4pool`**

Type
real(4)

Default
3.22e-7, 9.65e-9, 2.12e-8, 6.43e-10

Specific soil respiration rate for the 4-pool submodel for each soil carbon pool (decomposable plant material, resistant plant material, biomass, humus).

`JULES_SOIL_BIOGEOCHEM::`**`bio_hum_cn`**

Type
real

Default
10.0

Parameter controlling ratio of C to N for BIO and HUM pools.

`JULES_SOIL_BIOGEOCHEM::`**`sorp`**

Type
real

Default
10.0

Parameter controlling the leaching of inorganic N through the soil profile. A factor of 1 means that in a timestep all the inorganic N is available for leaching. The default value of 10 means that 10% of inorganic N is available for leaching.

`JULES_SOIL_BIOGEOCHEM::`**`n_inorg_turnover`**

Type
real

Default
1.0

Parameter controlling the lifetime of the inorganic N pool. A value of 1 implies the whole pool will turnover in 360 days.

JULES_SOIL_BIOGEOCHEM::**tau_resp**

Type
real

Default
2.0

Parameter controlling decay of respiration with depth (m-1). Only used with layered soil carbon (*l_layeredc* = TRUE).

JULES_SOIL_BIOGEOCHEM::**diff_n_pft**

Type
real

Default
5.0

Parameter controlling the rate of re-filling of the available inorganic nitrogen pool (1/360 days). This parameter determines how quickly the inorganic nitrogen reaches the roots after the roots uptake from the soil around them. This should be quicker than the turnover rate of inorganic nitrogen. In addition, it has to be small compared with the triffid timestep (360/triffid_period) otherwise the available inorganic nitrogen becomes unstable. Hence the choice of the default value 5. Only used with layered soil carbon and nitrogen scheme (*l_layeredc* = TRUE and *l_nitrogen* = TRUE). When *l_trif_eq* = TRUE or *diff_n_pft* is greater than (0.5 * 360 / *triffid_period*) then all of the inorganic nitrogen pool is deemed to be available.

Parameters for the 4-pool- or ECOSSE-based models (only used if soil_bgc_model = 2 or 3):

JULES_SOIL_BIOGEOCHEM::**tau_lit**

Type
real

Default
5.0

Parameter controlling the decay of litter with depth (m-1). With 4-pool, this is only used with layered soil carbon (*l_layeredc* = TRUE). With ECOSSE, this is only used with *plant_input_profile* = 2.

Methane parameters and switches. Can only be used with the single-pool and 4-pool models (soil_bgc_model = 1 or 2).

Warning: Some parameters may need to be re-tuned for different soil biogeochemistry models.

JULES_SOIL_BIOGEOCHEM::**l_ch4_tlayered**

Type
logical

Default
F

Switch to calculate methane emissions based on layered soil temperature.

TRUE
Methane emission is calculated from layered soil temperatures.

FALSE
Methane emission is calculated from top 1m average soil temperature (default).

`JULES_SOIL_BIOGEOCHEM::`**`l_ch4_interactive`**

Type
logical

Default
F

Switch to couple the methane emission into the carbon cycle. In order to use this the methane must be calculated from layered soil temperature (*`l_ch4_tlayered`* = TRUE).

TRUE
Methane flux is subtracted from soil carbon stocks.

FALSE
Methane emission is only diagnostic (default).

`JULES_SOIL_BIOGEOCHEM::`**`l_ch4_microbe`**

Type
logical

Default
F

Switch to enable the microbial methane production scheme (represents the dynamics of methanogens and a dissolved substrate pool). See Chadburn et al. (2020).

Note: This will only be applied to the methane production from your chosen *`ch4_substrate`*. The scheme has been calibrated with *`ch4_substrate`* = 1.

TRUE
Microbial dynamics simulated in methane scheme.

FALSE
No microbial dynamics, decomposition of substrate translates immediately to methane emissions.

`JULES_SOIL_BIOGEOCHEM::`**`ch4_substrate`**

Type
integer

Permitted
1, 2 or 3

Default
1

Choice of substrate for wetland methane. This controls the calculation method for the methane flux that is used to update soil carbon (only if *`l_ch4_interactive`* = TRUE) and to populate the variable fch4_wetl (seen by the atmospheric model in coupled mode).

Possible values are:

1. Using soil carbon as substrate (default).
2. Using NPP as substrate.
3. Using soil respiration as substrate.

This replaces the previous switch l_wetland_ch4_npp.

`JULES_SOIL_BIOGEOCHEM::`**`t0_ch4`**

Type
real

Default
273.15

Reference temperature for the Q10 function CH4 emission calculation

`JULES_SOIL_BIOGEOCHEM::`**`const_ch4_cs`**

Type
real

Default
7.41e-12

Scale factor for wetland CH4 emissions when soil carbon is taken as the substrate for ch4 emissions (*ch4_substrate* = 1)

Note: In the UM the recommended value depends on *l_triffid* as follows:

l_triffid = FALSE, *const_ch4_cs* = 5.41e-12
l_triffid = TRUE, *const_ch4_cs* = 5.41e-10

`JULES_SOIL_BIOGEOCHEM::`**`q10_ch4_cs`**

Type
real

Default
3.7

Q10 value for wetland CH4 emissions when soil carbon is taken as the substrate for ch4 emissions (*ch4_substrate* = 1)

`JULES_SOIL_BIOGEOCHEM::`**`const_ch4_npp`**

Type
real

Default
9.99e-3

Scale factor for wetland CH4 emissions when NPP is taken as the substrate for ch4 emissions (*ch4_substrate* = 2)

`JULES_SOIL_BIOGEOCHEM::`**`q10_ch4_npp`**

Type
real

Default
1.5

Q10 value for wetland CH4 emissions when npp is taken as the substrate for ch4 emissions (*ch4_substrate* = 2)

`JULES_SOIL_BIOGEOCHEM::`**`const_ch4_resps`**

Type
real

Default
4.36e-3

Scale factor for wetland CH4 emissions when soil respiration is taken as the substrate for ch4 emissions (*ch4_substrate* = 3)

`JULES_SOIL_BIOGEOCHEM::`**`q10_ch4_resps`**

Type
real

Default
1.5

Q10 value for wetland CH4 emissions when soil respiration is taken as the substrate for ch4 emissions (*`ch4_substrate`* = 3)

`JULES_SOIL_BIOGEOCHEM::`**`ch4_cpow`**

Type
real

Default
1.0

Power of soil carbon used to calculate methane emissions with soil carbon as substrate (*`ch4_substrate`* = 1). Methane production is calculated as cs^{ch4_cpow}. A value of 1.0 is default, but a value of 2/3 is consistent with an assumption that only the surfaces of the organic matter are accessible.

Note: *`const_ch4_cs`* will need retuning if this parameter is changed.

Methane parameters only used with layered soil temperatures (`l_ch4_tlayered` = TRUE).

`JULES_SOIL_BIOGEOCHEM::`**`tau_ch4`**

Type
real

Default
6.5

Exponent in the exponential decline of methane emissions with soil depth (m-1). This empirically represents methane oxidation/emission processes, which only allow a fraction of the methane produced in the soil to reach the atmosphere.

Methane parameters only used with microbial methane scheme (`l_ch4_microbe` = TRUE).

`JULES_SOIL_BIOGEOCHEM::`**`k2_ch4`**

Type
real

Default
0.01

Baseline methanogenic respiration rate (hr-1).

`JULES_SOIL_BIOGEOCHEM::`**`kd_ch4`**

Type
real

Default
0.0003

Baseline methanogenic mortality rate (hr-1).

`JULES_SOIL_BIOGEOCHEM::`**`rho_ch4`**

Type
real

Default
47.0

Factor in substrate limitation function (related to half saturation of substrate for methanogenic respiration) ((mgC/m3)-1).

`JULES_SOIL_BIOGEOCHEM::q10_mic_ch4`

Type
real

Default
4.3

Q10 factor for methanogens.

`JULES_SOIL_BIOGEOCHEM::cue_ch4`

Type
real

Default
0.03

Carbon use efficiency of methanogenic growth.

`JULES_SOIL_BIOGEOCHEM::mu_ch4`

Type
real

Default
0.00042

Threshold growth rate below which methanogens die (hr-1).

`JULES_SOIL_BIOGEOCHEM::frz_ch4`

Type
real

Default
0.5

Factor to reduce CH4 substrate production when soil is sufficiently frozen (only in microbial scheme).

`JULES_SOIL_BIOGEOCHEM::alpha_ch4`

Type
real

Default
0.001

Ratio between maintenance and growth respiration rates for methanogens.

`JULES_SOIL_BIOGEOCHEM::ev_ch4`

Type
real

Default
5.0

Timescale over which methanogenic traits adapt to temperature change (yr)

`JULES_SOIL_BIOGEOCHEM::q10_ev_ch4`

Type
real

Default
2.2

Q10 for temperature response of methanogenic traits under adaptation

See also:

Reference for microbial methane scheme:

- Chadburn, S. E. et al (2020), Modeled Microbial Dynamics Explain the Apparent Temperature Sensitivity of Wetland Methane Emissions. Global Biogeochemical Cycles, 34: e2020GB006678. https://doi.org/10.1029/2020GB006678

6.12 `jules_soil_ecosse.nml`

This file sets options and parameters for the ECOSSE model of soil biogeochemistry. It is used only if the ECOSSE model is chosen (*`soil_bgc_model`* = 3).

Warning: The ECOSSE model in JULES is still in development and is not fully functional in this version. The code is included to allow further development. Users should not try to use ECOSSE.

See also:

References:

- Smith et al., 2010, Estimating changes in Scottish soil carbon stocks using ECOSSE. I. Model description and uncertainties, Climate Research, 45: 179-192. (https://doi.org/10.3354/cr00899).
- Clark, D. B., Mercado, L. M., Sitch, S., Jones, C. D., Gedney, N., Best, M. J., Pryor, M., Rooney, G. G., Essery, R. L. H., Blyth, E., Boucher, O., Harding, R. J., Huntingford, C., and Cox, P. M.: The Joint UK Land Environment Simulator (JULES), model description – Part 2: Carbon fluxes and vegetation dynamics, Geosci. Model Dev., 4, 701–722, (https://doi.org/10.5194/gmd-4-701-2011), 2011.

6.12.1 `JULES_SOIL_ECOSSE` namelist members

`JULES_SOIL_ECOSSE::`**`l_soil_n`**

Type
logical

Default
T

Switch to include soil nitrogen in ECOSSE.

TRUE
Model soil carbon and nitrogen.

FALSE
Model only soil carbon.

`JULES_SOIL_ECOSSE::`**`l_match_layers`**

Type
logical

Default
T

Switch to match ECOSSE soil C and N layers to those for soil moisture.

TRUE
Use the same layering as for soil moisture. The number of soil carbon layers will equal *`sm_levels`*, with layer thicknesses given by *`dzsoil_io`*.

FALSE
The number of layers will be specified by *dim_cslayer* and the thicknesses by *dz_soilc_io*.

JULES_SOIL_ECOSSE::**dim_cslayer**

Type
integer

Permitted
>= 1

Default
None

The number of ECOSSE soil carbon layers. Not used if *l_match_layers* = TRUE. Despite the similar name, this parameter is unrelated to *sclayer_dim_name*.

JULES_SOIL_ECOSSE::**dz_soilc_io**

Type
real(dim_cslayer)

Permitted
> 0

Default
None

Thicknesses of the ECOSSE soil carbon layers (m). Not used if *l_match_layers* = TRUE. In most cases the total depth must equal that of the soil moisture layers (see *dzsoil_io*). The exception is the case of a single layer, bulk model (*dim_cslayer* = 1), for which dz_soilc(1) is interpreted as the representative or averaging depth for the bulk layer (e.g. the temperature of the bulk model is taken as the average over this depth).

JULES_SOIL_ECOSSE::**dt_soilc**

Type
REAL

Permitted
>= *timestep_len*

Default
timestep_len

The timestep length for ECOSSE (seconds). The main JULES timestep length *timestep_len* must be a multiple of this timestep length, so that ECOSSE is called on JULES timesteps. The TRIFFID timestep length *triffid_period* (converted to seconds) must also be a multiple of the ECOSSE timestep length.

JULES_SOIL_ECOSSE::**l_driver_ave**

Type
logical

Default
T

Switch controlling the averaging of the physical driving variables that are input to ECOSSE (e.g. soil temperature).

TRUE
Average the driving variables over the ECOSSE timestep length.

FALSE
Use instantaneous values of the driving variables at the time when ECOSSE is called.

JULES_SOIL_ECOSSE::**plant_input_profile**

Type
INTEGER

Permitted
1 or 2

Default
1

Switch for the vertical distribution of litterfall inputs of C and N to the soil.

Possible values are:

1. Fraction *pi_sfc_frac* of inputs are distributed uniformly in a surface layer of depth *pi_sfc_depth*, the remainder are distributed according to the distribution of roots.
2. Inputs decrease exponentially with depth with decay constant *tau_lit*.

`JULES_SOIL_ECOSSE::`**`pi_sfc_frac`**

Type
REAL

Default
0.3

Fraction of plant litterfall that is added to the surface soil layer of depth *pi_sfc_depth*. Only used if *plant_input_profile* = 1.

`JULES_SOIL_ECOSSE::`**`pi_sfc_depth`**

Type
REAL

Default
0.1

Depth of soil over which fraction *pi_sfc_frac* of plant litterfall is added (m). Only used if *plant_input_profile* = 1.

`JULES_SOIL_ECOSSE::`**`temp_modifier`**

Type
INTEGER

Permitted
1 or 2

Default
2

Switch for the form of the temperature rate modifier for decomposition.

1. Use a Q10 approach (Eqn. 65 of Clark et al. (2011))
2. Use the Smith et al. (2010) form of the modifier (Eqn.1 of Smith et al. (2010))

`JULES_SOIL_ECOSSE::`**`water_modifier`**

type
INTEGER

permitted
1 or 2

default
2

Switch for the form of the water rate modifier for decomposition and nitrification.

1. Use the Clark et al. (2011) form of the modifier (Eqn. 67 of Clark et al. (2011)). Note however that only the unfrozen water is considered here.
2. Use the Smith et al. (2010) form of the modifier.

JULES_SOIL_ECOSSE::**decomp_rate**

Type
REAL(4)

Default
3.22e-7, 9.65e-9, 2.12e-8, 6.43e-10

Rate constant for decomposition of each soil carbon pool (s^{-1}).

Note that these default values are also those for use with the 4-pool model (*kaps_4pool*).

JULES_SOIL_ECOSSE::**decomp_ph_rate_min**

Type
REAL

Default
0.2

Minimum allowed value of pH rate modifier for decomposition.

JULES_SOIL_ECOSSE::**decomp_ph_min**

Type
REAL

Permitted
decomp_ph_min <= decomp_ph_max

Default
1.0

Soil pH below which rate of decomposition is minimum.

JULES_SOIL_ECOSSE::**decomp_ph_max**

Type
REAL

Permitted
decomp_ph_max >= decomp_ph_min

Default
4.5

Soil pH above which rate of decomposition is maximum.

JULES_SOIL_ECOSSE::**decomp_temp_coeff_smith**

Type
REAL(3)

Default
47.9, 106.0, 18.3

Constants in the 4-pool from of the decomposition temperature modifier (*temp_modifier* = 2).

Note that these default values are also those harwired in the code for use with the 4-pool model (*soil_bgc_model* = 2, *l_q10* = F).

JULES_SOIL_ECOSSE::**decomp_wrate_min_smith**

Type
REAL

Default
0.2

Minimum allowed value of the water rate modifier for decomposition when the 4-pool form is used (*water_modifier* = 2).

`JULES_SOIL_ECOSSE::`**`decomp_wrate_min_clark`**

Type
REAL

Default
0.2

Minimum allowed value of the water rate modifier for decomposition when the Clark et al. (2011) form is used (*`water_modifier`* = 1).

Note that this default value is also that harwired in the code for use with the 4-pool model (*`soil_bgc_model`* = 2).

Parameters for ECOSSE that are only used if soil N is included (`l_soil_n` = TRUE).

`JULES_SOIL_ECOSSE::`**`l_decomp_slow`**

Type
logical

Default
T

Switch controlling how lack of nitrogen affects soil decomposition.

TRUE
Reduce the decomposition rate.

FALSE
Reduce the efficiency of decomposition, so that decomposition results in increased production of CO_2 and decreased production of further soil C. This is the approach used in the standalone version of ECOSSE.

`JULES_SOIL_ECOSSE::`**`depo_nit_frac`**

Type
REAL

Permitted
0 <= depo_nit_frac <= 1

Default
1.0

The fraction of nitrogen deposition that is aded to the soil nitrate pool. The complement is aded to the ammonium pool.

`JULES_SOIL_ECOSSE::`**`bacteria_min_frac`**

Type
REAL

Permitted
0 <= bacteria_min_frac <= 1

Default
0.2

The minimum fraction of the decomposer community that are bacteria.

`JULES_SOIL_ECOSSE::`**`bacteria_max_frac`**

Type
REAL

Permitted
bacteria_min_frac <= bacteria_max_frac <= 1

Default
0.5

The maximum fraction of the decomposer community that are bacteria.

`JULES_SOIL_ECOSSE::bacteria_min_frac_pH`

Type
REAL

Default
4.0

The soil pH at or below which the fraction of bacteria is at a minimum.

`JULES_SOIL_ECOSSE::bacteria_max_frac_pH`

Type
REAL

Permitted
bacteria_min_frac_pH <= bacteria_max_frac_pH

Default
5.5

The soil pH at or above which the fraction of bacteria is at a maximum.

`JULES_SOIL_ECOSSE::cn_bacteria`

Type
REAL

Default
5.5

The C:N ratio of soil bacteria.

`JULES_SOIL_ECOSSE::cn_fungi`

Type
REAL

Default
5.5

The C:N ratio of soil fungi.

`JULES_SOIL_ECOSSE::depth_nitrif`

Type
REAL

Default
0.25

Greatest depth at which nitrification and denitrification are allowed (m).

`JULES_SOIL_ECOSSE::nitrif_rate`

Type
REAL

Default
9.921e-7

Rate constant for nitrification (s^{-1}).

`JULES_SOIL_ECOSSE::nitrif_wrate_min`

Type
REAL

Default
0.6

Minimum allowed value of the water rate modifier for nitrification when 4-pool form is used. Only used if `water_modifier` = 2.).

`JULES_SOIL_ECOSSE::`**`nitrif_max_factor`**

Type
REAL

Default
0.1

Shape factor in rate modifier for nitrification (kg m^{-3}).

`JULES_SOIL_ECOSSE::`**`nitrif_frac_n2o_fc`**

Type
REAL

Permitted
0 <= nitrif_frac_n2o_fc <= 1

Default
0.02

Fraction of nitrification lost as N_2O by partial nitrification at field capacity.

`JULES_SOIL_ECOSSE::`**`nitrif_frac_gas`**

Type
REAL

Permitted
0 <= nitrif_frac_gas <= 1

Default
0.02

Fraction of nitrification lost as gas through full nitrification.

`JULES_SOIL_ECOSSE::`**`nitrif_frac_no`**

Type
REAL

Permitted
0 <= nitrif_frac_no <= 1

Default
0.4

Fraction of nitrification gas loss through full nitrification that is NO.

`JULES_SOIL_ECOSSE::`**`denit50`**

Type
REAL

Default
0.033

Amount of nitrate at which denitrification rate is 50% of the potential rate (kg m^{-3}).

`JULES_SOIL_ECOSSE::`**`denit_bio_factor`**

Type
REAL

Default
0.005

Factor in denitrification calculation to convert flux of CO_2 into a representation of biological activity (m^2 kg^{-1}).

JULES_SOIL_ECOSSE::**denit_frac_n2_fc**

Type
REAL

Permitted
0 <= denit_frac_n2_fc <= 1

Default
0.55

Proportion of denitrified N that becomes N_2 when soil moisture is at field capacity.

JULES_SOIL_ECOSSE::**denit_nitrate_equal**

Type
REAL

Default
0.4

Amount of N in soil nitrate at which denitrified N is released as equal amounts of N_2 and $N_{2`O (kg m:sup:}$-3`).

JULES_SOIL_ECOSSE::**denit_water_coeff**

Type
REAL(3)

Default
0.62, 0.38, 1.74

Constants describing water modifier for denitrification.

JULES_SOIL_ECOSSE::**amm_leach_min**

Type
REAL

Default
0.02

Minimum allowed amount of N in soil ammonium after leaching (kg m^{-3}).

JULES_SOIL_ECOSSE::**n_inorg_max_conc**

Type
REAL

Default
-1.0

Maximum-allowed concentration of inorganic N in a layer (kg m^{-3}). A value less than zero means no maximum concentration is imposed.

6.13 jules_deposition.nml

This file contains options and parameters for modelling of dry deposition of atmospheric trace constituents. It contains a variable number of namelists depending on the required model configuration. The namelist *JULES_DEPOSITION* is always required, and one or more instances of the namelist *JULES_DEPOSITION_SPECIES* is required if dry deposition has been selected.

Warning: Atmospheric deposition in JULES is still in development and is far from fully functional in this version - the code is included to allow further development. Users should not try to activate the deposition code: leave *l_deposition* as FALSE.

6.13.1 `JULES_DEPOSITION` namelist members

`JULES_DEPOSITION::`**`l_deposition`**

Type
logical

Default
FALSE

Switch to activate deposition code in JULES.

TRUE
Model deposition in JULES.

FALSE
Do not model deposition in JULES.

Only used if `l_deposition` = TRUE.

`JULES_DEPOSITION::`**`dry_dep_model`**

Type
integer

Permitted
1

Default
none

Choice for model of dry deposition.

Possible values are:

1. Parameterisation based on that found in UKCA (but now implemented in JULES).

`JULES_DEPOSITION::`**`l_deposition_flux`**

Type
logical

Default
FALSE

Switch for calculation of deposition fluxes as opposed to deposition velocities.

TRUE
Calculate deposition fluxes. This requires that the concentrations of atmopsheric tracer species are provided as prescribed data (see *List of supported variables*).

FALSE
Calculate deposition velocities.

Only used if `l_deposition` = TRUE and `dry_dep_model` = 1 (UKCA).

`JULES_DEPOSITION::`**`ndry_dep_species`**

Type
integer

Permitted
>= 1

Default
none

Number of species for which dry deposition is calculated.

`JULES_DEPOSITION::`**`l_ukca_ddep_lev1`**

Type
logical

Default
FALSE

Switch controlling which atmospheric levels are used for dry deposition.

TRUE
Deposition occurs only from the lowest atmospheric level.

FALSE
Deposition occurs from all levels within the atmospheric boundary layer.

`JULES_DEPOSITION::`**`tundra_s_limit`**

Type
real

Default
none

Latitude of southern limit of tundra (degrees). This is used to alter the calculation of deposition of certain species in the tundra region (actually for all points north of this limit). Only used if the list of species (see *`dep_species_name_io`*) includes one or more of 'CO', 'NO2', 'O3', 'PAN', 'PPAN', 'MPAN' or 'ONITU'.

`JULES_DEPOSITION::`**`dzl_const`**

Type
real

Default
none

Constant value for separation of boundary layer levels (m). All layer thicknesses are set to this value. This is used as a simple way to prescribe the layer thicknesses in standalone mode. This value can be overriden by prescribed data - see *`JULES_PRESCRIBED`*. This can be considered as the representative depth for tracer concentration and the depth over which the deposition flux is removed.

6.13.2 Notes on the `JULES_DEPOSITION` namelist

The height of the atmospheric boundary layer is required and is set to a default constant value of 1000 m. This value can be overridden via the namelist variable *`bl_height`*, or can be prescribed (i.e. allowed to vary in time and/or space) by providing `bl_height` as prescribed data (see *List of supported variables*).

The number of model levels in the boundary layer is required and is set to a default balues of 1. This can be overridden via the namelist variable *`bl_levels`*. The number of levels is only used to communicate with the atmospheric model (e.g. UKCA).

The separation of the model levels in the boundary layer is required and is set to a constant value via *`dzl_const`*. The separation can be prescribed (i.e. allowed to vary in time and/or space) by providing `level_separation` as prescribed data (see *List of supported variables*). This can be considered as the representative depth for tracer concentration and the depth over which the deposition flux is removed.

If deposition fluxes (rather than deposition velocities) are to be calculated (see *`l_deposition_flux`*), the concentrations of atmospheric tracer species need to be prescribed (see `tracer_field` in *List of supported variables*).

6.13.3 `JULES_DEPOSITION_SPECIES` namelist members

This namelist should occur *ndry_dep_species* times, with each occurence containing parameters for an atmospheric tracer species that is to be considered in dry deposition.

`JULES_DEPOSITION_SPECIES::`**`dep_species_name_io`**

> **Type**
> character
>
> **Default**
> none
>
> Name of an atmospheric tracer species to be included in deposition modelling.

`JULES_DEPOSITION_SPECIES::`**`diffusion_coeff_io`**

> **Type**
> real
>
> **Default**
> none
>
> Diffusion coefficient (m^{-2} s^{-1}).

`JULES_DEPOSITION_SPECIES::`**`rsurf_std_io`**

> **Type**
> real(ntype)
>
> **Default**
> none
>
> Standard value of surface resistance for each surface type (s m^{-1}).

Only used if `dep_species_name_io` = `NO2`, `O3`, `PAN`, `PPAN`, `MPAN` or `ONITU`. Values provided for other species will be ignored.

`JULES_DEPOSITION_SPECIES::`**`diffusion_corr_io`**

> **Type**
> real
>
> **Default**
> none
>
> Diffusion correction factor for stomatal resistance, accounting for the different diffusivities of water and other species (dimensionless).

Only used if `dep_species_name_io` = `CO`, `NO2`, `O3`, `PAN`, `PPAN`, `MPAN` or `ONITU`. Values provided for other species will be ignored.

`JULES_DEPOSITION_SPECIES::`**`r_tundra_io`**

> **Type**
> real
>
> **Default**
> none
>
> Surface resistance used in tundra region (s m^{-1}).

Only used if `dep_species_name_io` = `CH4`.

JULES_DEPOSITION_SPECIES::**ch4_scaling_io**

Type
real

Default
none

Scaling applied to methane soil uptake (dimensionless). (Originally this was used to match the value from the IPCC Third Assessment Report.)

JULES_DEPOSITION_SPECIES::**ch4dd_tundra_io**

Type
real(4)

Default
none

Coefficients of cubic polynomial relating methane loss for tundra to temperature. Flux is in units of ug (CH_4) m^{-2} s^{-1}.

Only used if dep_species_name_io = H2.

JULES_DEPOSITION_SPECIES::**h2dd_c_io**

Type
real(ntype)

Default
none

Constant in quadratic function relating hydrogen deposition to soil moisture (s m^{-1}).

JULES_DEPOSITION_SPECIES::**h2dd_m_io**

Type
real(ntype)

Default
none

Coefficient of first order term in quadratic function relating hydrogen deposition to soil moisture (s m^{-1}).

JULES_DEPOSITION_SPECIES::**h2dd_q_io**

Type
real(ntype)

Default
none

Coefficient of second order term in quadratic function relating hydrogen deposition to soil moisture (s m^{-1}).

Only used if dep_species_name_io = HNO3, HONO2 or ISON.

JULES_DEPOSITION_SPECIES::**dd_ice_coeff**

Type
real(3)

Default
none

Coefficients in quadratic function relating dry deposition over ice to temperature.

Only used if `dep_species_name_io` = O3.

`JULES_DEPOSITION_SPECIES::`**`cuticle_o3_io`**

Type
real

Default
none

Constant for calculation of cuticular resistance for ozone (s m^{-1}).

`JULES_DEPOSITION_SPECIES::`**`r_wet_soil_o3_io`**

Type
real

Default
none

Wet soil surface resistance for ozone (s m^{-1}).

6.14 `jules_snow.nml`

This file sets the snow options and parameters. It contains one namelist called *`JULES_SNOW`*.

6.14.1 `JULES_SNOW` namelist members

HCTN30 refers to Hadley Centre technical note 30, available from the Met Office Library.

`JULES_SNOW::`**`nsmax`**

Type
integer

Permitted
>= 0

Default
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.

`JULES_SNOW::`**`l_snowdep_surf`**

Type
logical

Default
F

TRUE
Use equivalent canopy snow depth for surface calculations on surface tiles with a snow canopy.

FALSE
No effect.

`JULES_SNOW::frac_snow_subl_melt`

Type
integer

Permitted
0 or 1

Default
0

Switch for use of snow-cover fraction in the calculation of sublimation and melting.

0. Off
1. On

`JULES_SNOW::graupel_options`

Type
integer

Permitted
0 or 1 or 2

Default
0

Switch for treatment of graupel in the snow scheme

0. Include graupel as snowfall
1. Ignore graupel in the surface snowfall
2. Treat graupel separately

Always "Include graupel as snowfall" (option 0) in standalone JULES because separate snow and graupel driving data are not available. If graupel is included in the UM surface snowfall diagnostic then JULES can either include this graupel as snow in the surface scheme (option 0), ignore this graupel completely, thereby breaking conservation of water and energy in the coupled land-atmosphere model (option 1) or treat graupel seperately (currently this only means allowing graupel to fall straight through the canopy)

`JULES_SNOW::dzsnow`

Type
real(nsmax)

Default
None

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 below.

`dzsnow` gives the thickness of each layer when it is not the bottom layer.

For the top layer, 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.

`JULES_SNOW::`**`cansnowpft`**

Type
logical(npft)

Default
F

Flag indicating whether snow can be held under the canopy of each PFT.

Only used if *`can_model`* = 4.

The model of snow under the canopy is currently only suitable for coniferous trees.

TRUE
Snow can be held under the canopy.

FALSE
Snow cannot be held under the canopy.

Radiation parameters

`JULES_SNOW::`**`r0`**

Type
real

Default
50.0

Grain size for fresh snow (µm).

Only used if *`l_snow_albedo`* = TRUE. See HCTN30 Eq.15.

`JULES_SNOW::`**`rmax`**

Type
real

Default
2000.0

Maximum snow grain size (µm).

Only used if *`l_snow_albedo`* = TRUE. See HCTN30 p4.

`JULES_SNOW::`**`snow_ggr`**

Type
real(3)

Default
0.6, 0.06, 0.23e6

Snow grain area growth rates (µm^2 s^{-1}).

Only used if *`l_snow_albedo`* = TRUE. See HCTN30 Eq.16.

The 3 values are for melting snow, cold fresh snow and cold aged snow respectively.

`JULES_SNOW::`**`amax`**

Type
real(2)

Default
0.98, 0.7

Maximum albedo for fresh snow.

Only used if *`l_snow_albedo`* or *`l_elev_land_ice`* are true

Values 1 and 2 are for VIS and NIR wavebands respectively.

JULES_SNOW::**aicemax**

Type
real(2)

Default
0.78, 0.36

Maximum albedo for bare ice

Only used if `l_elev_land_ice` = TRUE. See also *rho_firn_albedo*

Values 1 and 2 are for VIS and NIR wavebands respectively.

JULES_SNOW::**maskd**

Type
real

Default
50.0

Used in exponent of equation weighting snow-covered and snow-free albedo.

JULES_SNOW::**dtland**

Type
real

Default
2.0

Degrees Celsius below zero at which snow albedo equals cold deep snow albedo.

Only used if `l_snow_albedo` = FALSE. This is 2.0 in HCTN30 Eq4.

JULES_SNOW::**kland_numerator**

Type
real

Default
0.3

Used in snow-ageing effect on albedo.

Only used if `l_snow_albedo` = FALSE.

Must not be zero.

`kland` is computed by dividing this value by `dtland` - see HCTN30 Eq4.

JULES_SNOW::**can_clump**

Type
real(npft)

Default
MDI

Clumping parameter for snow on the canopy in calculation of albedo.

Only used if `can_model` = 4, `cansnowpft` = TRUE on that surface tile and `l_embedded_snow` = TRUE.

The model of snow under the canopy is currently only suitable for coniferous trees.

The inverse of this parameter specifies the fraction of the canopy over which snow is distributed when calculating the albedo.

JULES_SNOW::**n_lai_exposed**

Type
real(npft)

Default
MDI

LAI distribution parameter for calculation of snow albedo.

A power-law distribution of leaf area density is assumed within the canopy for calculating masking of snow by vegetation using the embedded scheme. Larger values imply greater densities toward the base of the canopy.

Only used if *l_embedded_snow* = TRUE.

JULES_SNOW::**lai_alb_lim_sn**

Type
real(npft)

Default
MDI

Minimum LAI in calculation of albedo in the presence of snow.

A minimum albedo is imposed when calculating the albedo of plant canopies (historically 0.5). This parameter allows it to be set for each PFT in the presence of snow. A separate variable, *lai_alb_lim_io* is used in the absence of snow.

Other snow parameters

JULES_SNOW::**rho_snow_const**

Type
real

Default
250.0

Constant density of lying snow (kg m^{-3}).

This value is used if *nsmax* = 0, in which case all snow is modelled as a single layer of constant density. If *nsmax* > 0, snow density is prognostic.

JULES_SNOW::**rho_snow_fresh**

Type
real

Default
100.0

Density of fresh snow (kg m^{-3}).

This value is only used if *nsmax* > 0.

JULES_SNOW::**rho_firn_albedo**

Type
real

Default
550.0

If *l_elev_land_ice* = TRUE, this is the threshold density (as measured over the ~top 10cm, depending on how the dzsnow layers are specified) at which the grain-size calculation of prognostic snow albedo will switch to one dependent on the surface density of the snowpack. Albedo is linearly scaled between *amax* for *rho_snow_const* and *aicemax* for rho_ice=917 kg/m^3.

JULES_SNOW::**snow_hcon**

Type
real

Default
0.265

Thermal conductivity of lying snow (W m^{-1} K^{-1}).

See HCTN30 Eq.42.

`JULES_SNOW::`**`snow_hcap`**

Type
real

Default
0.63e6

Thermal capacity of lying snow (J K^{-1} m^{-3}).

`JULES_SNOW::`**`snowliqcap`**

Type
real

Default
0.05

Liquid water holding capacity of lying snow, as a fraction of snow mass.

Only used if *`nsmax`* > 0.

`JULES_SNOW::`**`snowinterceptfact`**

Type
real

Default
0.7

Constant in relationship between mass of intercepted snow and snowfall rate.

Only used if *`can_model`* = 4.

`JULES_SNOW::`**`snowloadlai`**

Type
real

Default
4.4

Ratio of maximum canopy snow load to leaf area index (kg m^{-2}).

Only used if *`can_model`* = 4.

`JULES_SNOW::`**`snowunloadfact`**

Type
real

Default
0.4

Constant in relationship between canopy snow unloading and canopy snow melt rate.

Only used if *`can_model`* = 4.

`JULES_SNOW::`**`unload_rate_cnst`**

Type
real(npft)

Default
MDI

Constant term in the background unloading rate for snow on the canopy.

Only used if *can_model* = 4 and *cansnowpft* = TRUE on that surface tile.

JULES_SNOW::**unload_rate_u**

Type
real(npft)

Default
MDI

Term proportional to wind speed in unloading rate for snow on the canopy.

Only used if *can_model* = 4 and *cansnowpft* = TRUE on that surface tile.

JULES_SNOW::**i_snow_cond_parm**

Type
integer

Permitted
0 or 1

Default
MDI

Scheme used to calculate the conductivity of snow

Two parametrizations of snow conductivity are available taken from the papers of *Yen (1981)* and *Calonne et al. (2011)*.

Only used if *nsmax* > 0.

0	Yen (1981)
1	Calonne et al. (2011)

JULES_SNOW::**l_et_metamorph**

Type
logical

Default
F

TRUE
Include the effect of thermal metamorphism on the snow density.

FALSE
No effect.

This parametrization follows the form used by eg. Dutra et al. (2010)

JULES_SNOW::**l_snow_infilt**

Type
logical

Default
F

TRUE
Pass rainfall and melting from the canopy to the snowpack as infiltration.

FALSE
No effect.

JULES_SNOW::**l_snow_nocan_hc**

Type
logical

Default
F

TRUE
Do not include the canopy heat capacity in the surface energy balance at the top of the snow pack on surface tiles without a canopy snow model.

FALSE
The canopy heat capacity is include in the surface energy balance at the top of the snow pack.

JULES_SNOW::**a_snow_et**

Type
real

Default
MDI

Constant in parametrization of thermal metamorphism.

Only used if *l_et_metamorph* = TRUE.

JULES_SNOW::**b_snow_et**

Type
real

Default
MDI

Constant in parametrization of thermal metamorphism.

Only used if *l_et_metamorph* = TRUE.

JULES_SNOW::**c_snow_et**

Type
real

Default
MDI

Constant in parametrization of thermal metamorphism.

Only used if *l_et_metamorph* = TRUE.

JULES_SNOW::**rho_snow_et_crit**

Type
real

Default
MDI

Critical density in parametrization of thermal metamorphism.

Only used if *l_et_metamorph* = TRUE.

JULES_SNOW::**i_grain_growth_opt**

Type
integer

Permitted
0 or 1

Default
0

Scheme used to calculate the rate of growth of snow grains.

Setting this to 0 invokes the original scheme based on Marshall (1989), with no dependence of the rate of growth of small grains on the temperature.

Setting it to 1 invokes the scheme for growth of snow grains proposed by Taillandier et al. (2007) for equitemperature metamorphism. This is significantly slower than the default scheme at low temperatures.

`JULES_SNOW::`**`i_relayer_opt`**

Type
integer

Permitted
0 or 1

Default
0

Scheme used to relayer the snowpack. Setting the option to 0 invokes the original scheme with relayering of the grain size involving the grain size itself, while setting it to 1 causes the relayering to be done using the inverse of the grain size. This is more consistent with conserving the SSA, though full conservation would require mass weighting to be invoked during regridding.

Only used if *`nsmax`* > 0.

`JULES_SNOW::`**`i_basal_melting_opt`**

Type
integer

Permitted
0 or 1

Default
0

Option to treat basal melting of the snow pack. When snow falls on warm ground, it will melt from the base of the snowpack, where the temperature of the snow will rise to the melting point. The 0-layer snow scheme, which is used for thin snow even when the multilayer scheme is selected, did not represent this process and included only melting at the surface. This option allows basal melting to be omitted if it is set to the defaut value of 0, but offers an alternative setting of 1, which results in basal melting taking place instantaneously if the temperature of the first soil layer is above freezing, until the snow is removed or the temperature of soil layer is reduced to freezing.

6.14.2 Example of the evolution of snow layer thickness

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

Snow depth (m)	Number of layers	Layer thickness, uppermost layer first (m)	Comments
`d < 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 <= d < 0.2`	1	Total snow depth	The single layer grows until it is twice as thick as `dzsnow(1)`.
`0.2 <= d < 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.
`>= 0.4`	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.

6.14.3 `JULES_SNOW` references

- Calonne, N., Flin, F., Morin, S., Lesaffre, B., du Roscoat, S. Rolland, and Geindreau, C. (2011), Numerical and experimental investigations of the effective thermal conductivity of snow, Geophys. Res. Lett., 38, L23501, https://doi.org/10.1029/2011GL049234.
- Yen, Y.-C. (1981). Review of thermal properties of snow, ice and sea ice. Cold Regions Research and Engineering Laboratory (CRREL) Report 81-10. https://hdl.handle.net/11681/9469

6.15 `jules_rivers.nml`

This file sets the river routing options. It contains two namelists called *`JULES_RIVERS`* and *`JULES_OVERBANK`*.

River routing introduces two more grids to a JULES run: the river routing input grid and the river routing model grid. The river routing input grid must always be specified as a 2D grid in *`JULES_RIVERS_PROPS`*. This is not required to be identical to the input or the model grid. Internally the model compresses this to the river routing model grid, which is a 1D grid with length `np_rivers`, which is the number of valid routing points in the river routing input grid. All river routing output will be on the river routing model grid, or will be regridded to the model grid.

Note: The river routing code in JULES is still in development. Users should ensure that results are as expected, and provide feedback where deficiencies are identified.

6.15.1 `JULES_RIVERS` namelist members

`JULES_RIVERS::`**`l_rivers`**

Type
logical

Default
F

Switch for enabling river routing.

TRUE
Use the river routing algorithm specified by *`i_river_vn`* to route runoff along river pathways.

FALSE
: No river routing.

`JULES_RIVERS::`**`i_river_vn`**

Type
: integer

Default
: None

Switch to select the river routing algorithm to use for river routing.

1
: Use a UM-coupled JULES implementation of the TRIP model (see Oki et al. 1999). This value is not allowed in standalone JULES

2
: Use a standalone JULES implementation of the RFM kinematic wave model (see Dadson and Bell 2010, Bell et al. 2007).

3
: Use a standalone JULES implementation of the TRIP model (see Oki et al. 1999).

`JULES_RIVERS::`**`nstep_rivers`**

Type
: integer

Permitted
: > 0

Default
: None

The number of model timesteps per routing timestep.

For example, *`nstep_rivers`* = 5 means that runoff will be accumulated for 5 model timesteps before being routed on the 5th timestep.

Warning: The river routing parameter values can be highly dependent on model resolution, so care is required by the user to ensure that appropriate values are selected, tested and adjusted as required.

Suggested values for global and high-resolution runs are listed below, however these should be treated as a starting point only.

RFM parameters - used if `i_river_vn` = 2

`JULES_RIVERS::`**`a_thresh`**

Type
: integer

Default
: None

Suggested
: 1 (spatial resolution coarser than 20 km gridcells), ~10 (high-resolution)

The threshold drainage area (specified in number of cells) draining to a gridbox, above which the grid cell is considered to be a river point (see a_T in Bell et al. 2007:541).

Remaining points are treated as land (drainage area = 0) or sea (drainage area < 0). See Bell et al. (2007).

`JULES_RIVERS::`**`cland`**

Type
: real

Permitted
> 0

Default
None

Suggested
0.20 m/s (global), 0.40 m/s (1 km resolution, Bell et al. 2007)

The land wave speed (kinematic wave speed for surface flow in a land grid box on the river routing grid, m s^{-1}). This is the speed at which water moves through surface soil in a non-river grid cell (even without major rivers, there are always minor water courses so these cells do still contribute flow to neighbouring cells).

`JULES_RIVERS::criver`

Type
real

Permitted
> 0

Default
None

Suggested
0.62 m/s (global), 0.50 m/s (1 km resolution, Bell et al. 2007)

The river wave speed (kinematic wave speed for surface flow in a river grid box on the river routing grid, m s^{-1}). This value should be close to the *rivers_speed* used by TRIP, but not identical because RFM makes different assumptions about e.g. meandering.

`JULES_RIVERS::cbland`

Type
real

Permitted
> 0

Default
None

Suggested
<= *cland*. 0.10 m/s (global), 0.05 m/s (1 km resolution, Bell et al. 2007)

The subsurface land wave speed (kinematic wave speed for subsurface flow in a land grid box on the river routing grid, m s^{-1}).

`JULES_RIVERS::cbriver`

Type
real

Permitted
> 0

Default
None

Suggested
<= *criver*. 0.15 m/s (global), 0.05 m/s (1 km resolution, Bell et al. 2007)

The subsurface river wave speed (kinematic wave speed for subsurface flow in a river grid box on the river routing grid, m s^{-1}).

`JULES_RIVERS::retl`

Type
real

Permitted
-1 to 1

Default
None

Suggested
0.005 (1 km resolution, Bell et al. 2007)

The (resolution dependent) land return flow fraction. Bell et al. (2007:Table1) suggested value 0.005. On non-river grid cells in the land mask: if retl>0 then fraction retl of the subsurface flow moves to the surface per routing timestep; if retl<0 then fraction retl of the surface flow moves to the subsurface per routing timestep.

`JULES_RIVERS::`**`retr`**

Type
real

Permitted
-1 to 1

Default
None

Suggested
0.005 (1 km resolution, Bell et al. 2007)

The (resolution dependent) river return flow fraction. On river grid cells in the land mask: if retr>0 then fraction retr of the subsurface flow moves to the surface per routing timestep; if retr<0 then fraction retr of the surface flow moves to the subsurface per routing timestep.

`JULES_RIVERS::`**`runoff_factor`**

Type
real

Permitted
> 0

Default
None

Values !=1.0 are generally used to correct biases in precipitation when the model is forced with observed data **It is highly recommended that this is set to 1.0 (i.e. no runoff adjustment).**

TRIP parameters - used if `i_river_vn = 1,3`

`JULES_RIVERS::`**`rivers_speed`**

Type
real

Permitted
> 0

Default
None

The effective river velocity (m s^{-1}). See Oki et al. (1999). `rivers_speed` should equal (river flow velocity / `rivers_meander`). A value of 0.4 can be used, while Oki et al. (1999) used a value of 0.5.

`JULES_RIVERS::`**`rivers_meander`**

Type
real

Permitted
> 0

Default
None

The ratio of the actual to calculated river lengths in a river routing gridbox. See Oki et al. (1999). Oki & Sud (1998) called this the Meandering Ratio r_M and suggested an average global value of 1.4.

See also:

References:

- Arora VK & Boer GJ (2012). A variable velocity flow routing algorithm for GCMs. Journal of Geophysical Research D 104:30965-30979.
- Bell, V.A. et al. (2007) Development of a high resolution grid-based river flow model for use with regional climate model output. Hydrology and Earth System Sciences. 11 532-549
- Dadson, S.J. and Bell, V.A. (2010) Comparison of Grid-2-Grid and TRIP runoff routing schemes. Centre for Ecology & Hydrology Internal Report http://nora.nerc.ac.uk/10890/1/dadson_etal_2010_g2gtrip.pdf
- Dadson S.J. et al. (2011) Evaluation of a grid-based river flow model configured for use in a regional climate model. Journal of Hydrology. 411 238-250
- Falloon, P.D. et al (2007) New global river routing scheme in the Unified Model. Hadley Centre Technical Note 72, available from the Met Office Library.
- Jones R., Dadson, S. and Bell, V.A. (2007) Report on European grid-based river-flow modelling for application to Regional Climate Models. Met Office Hadley Centre deliverable report.
- Oki, T. and Sud, Y.C. (1998) Design of Total Runoff Integrating Pathways (TRIP)—A Global River Channel Network. Earth Interactions, 2: 1-37.
- Oki, T., et al (1999) Assessment of annual runoff from land surface models using Total Runoff Integrating Pathways (TRIP). Journal of the Meteorological Society of Japan. 77 235-255

6.15.2 `JULES_OVERBANK` namelist members

Warning: The overbank inundation parameter values can be highly dependent on model resolution, so care is required by the user to ensure that appropriate values are selected, tested and adjusted as required.

Suggested values for global and high-resolution runs are listed below, however these should be treated as a starting point only.

`JULES_OVERBANK::`**`l_riv_overbank`**

Type
logical

Default
F

Switch for enabling river overbank inundation. Only used if *`l_rivers`* is TRUE.

TRUE
Calculate frac_fplain_lp, i.e. overbank inundation area as a fraction of gridcell area.

FALSE
No overbank inundation calculations

Note: If *`l_riv_overbank`* = FALSE, no further variables are needed from this namelist.

`JULES_OVERBANK::`**`l_riv_hypsometry`**

Type
logical

Default
F

Switch for enabling use of a hypsometric integral calculation.

TRUE
Calculate inundated area from a hypsometric integral based on a lognormal area-altitude distribution (**recommended**).

FALSE
Estimate inundated area from simple river width scaling, ignoring topography (only to be used for testing).

River depth allometry (used if `l_riv_hypsometry` is TRUE or `use_rosgen` is TRUE)

Allometry is: (DEPTH in m) = *riv_c* * ((SURFACE RIVER INFLOW in m3 s^{-1}) ^ *riv_f*) (Leopold & Maddock 1953:eqn2)

`JULES_OVERBANK::`**`riv_c`**

Type
real

Default
none

Permitted
>=0 and <=(1/*riv_a*)

Suggested
0.27 (global, from Andreadis et al. 2013)

Coefficient in the allometry for river depth (units are (m / ((m3/s)^riv_f)), i.e. dependent on the value of riv_f)

`JULES_OVERBANK::`**`riv_f`**

Type
real

Default
none

Permitted
>=0 and <=(1-*riv_b*)

Suggested
0.30 (global, from Andreadis et al. 2013)

Exponent in the allometry for river depth (dimensionless)

River width scaling (used if `l_riv_hypsometry` is FALSE)

River width allometry

Allometry is: (WIDTH in m) = *riv_a* * ((SURFACE RIVER INFLOW in m3 s^{-1}) ^ *riv_b*) (Leopold & Maddock 1953:eqn1)

`JULES_OVERBANK::`**`riv_a`**

Type
real

Default
none

Permitted
>=0 and <=(1/*riv_c*)

Suggested
7.20 (global, from Andreadis et al. 2013)

Coefficient in the allometry for river width (units are (m / ((m3/s)^riv_b)), i.e. dependent on the value of riv_b)

`JULES_OVERBANK::riv_b`

Type
real

Default
none

Permitted
>=0 and <=(1-*riv_f*)

Suggested
0.50 (global, from Andreadis et al. 2013)

Exponent in the allometry for river width (dimensionless)

`JULES_OVERBANK::use_rosgen`

Type
logical

Default
F

Switch for applying the Rosgen entrenchment ratio approach to estimate river width

TRUE
When inflow rates are lower than bankfull flow, river width is calculated from the River width allometry (above). However, when higher than bankfull flow, river width is constrained so that when river depth = 2 x bankfull depth then width = *ent_ratio* * bankfull width.

FALSE
River width follows the allometry specified above whatever the inflow rate.

Bankfull flow allometry (used if `use_rosgen` is TRUE) (Rosgen 1994)

Allometry is: (BANKFULL DISCHARGE RATE QBF in m3 s^{-1}) = *coef_b* * ((CONTRIBUTING AREA in km2) ^ *exp_c*) (see e.g. Andreadis et al. 2013)

`JULES_OVERBANK::coef_b`

Type
real

Default
none

Suggested
0.08 (for "several drainages in western Washington State, USA", Cragun 2005)

Coefficient in the allometry for bankfull flow (see Sen 2018:eqn3.33).

`JULES_OVERBANK::exp_c`

Type
real

Default
none

Suggested
0.95 (for "several drainages in western Washington State, USA", Cragun 2005)

Exponent in the allometry for bankfull flow (see Sen 2018:eqn3.33).

`JULES_OVERBANK::ent_ratio`

Type
real

Default
none

The Rosgen entrenchment ratio (single value for all water courses in the simulation): when river depth = 2 x bankfull depth then width = *ent_ratio* * bankfull width (i.e. *ent_ratio* can be used to specify how wide floodplains are allowed to be).

See also:

References:

- Andreadis KM, Schumann GJ & Pavelsky T (2013). A simple global river bankfull width and depth database. Water Resources Research 49:7164-7168
- Cragun WS (2005). Discharge-Area relations from Selected Drainages on the Colorado Plateau: A GIS Application. Utah State University, http://hydrology.usu.edu/giswr/archive05/scragun/termproject/
- Leopold LB & Maddock T (1953). The Hydraulic Geometry of Stream Channels and Some Physiographic Implications. United States Geological Survey Professional Papers 252:1-57
- Rosgen DL (1994). A classification of natural rivers. Catena 22:169-199.
- Sen Z (2018). Flood Modeling, Prediction and Mitigation. Springer.

6.16 `jules_water_resources.nml`

This file sets options for water resource modelling. It contains a single namelist called *JULES_WATER_RESOURCES*.

Warning: The water resource code in JULES is still in development. It should only be used by the developers.

6.16.1 `JULES_WATER_RESOURCES` namelist members

`JULES_WATER_RESOURCES::l_water_resources`

Type
logical

Default
F

Switch to enable modelling of water resources.

TRUE
Water resources are modelled. This also requires that river routing is selected (*l_rivers* = TRUE).

FALSE
No water resources. In this case no further values from this namelist are required.

`JULES_WATER_RESOURCES::nstep_water_res`

Type
integer

Permitted
> 0

Default
none

The number of model timesteps per water resource timestep. (The main model timestep is given by `timestep_len`.)

For example, `nstep_water_res` = 12 means that demands for water will be accumulated over 12 model timesteps before the water resource code is called on the 12th timestep.

The water resource and river routing models must be in synchrony (i.e. be called on the same timesteps).

Switches that control which water sectors are considered.

`JULES_WATER_RESOURCES::l_water_domestic`

Type
logical

Default
F

Switch for modelling of water for domestic use.

TRUE
Consider demand for water for domestic use. This requires that the domestic demand is prescribed as an input to the model (see *prescribed_data.nml*).

FALSE
Do not consider domestic demand.

`JULES_WATER_RESOURCES::l_water_environment`

Type
logical

Default
F

Switch for modelling of water for environmental flow requirements.

TRUE
Consider demand for water for environmental flows.

FALSE
Do not consider environmental demand.

`JULES_WATER_RESOURCES::l_water_industry`

Type
logical

Default
F

Switch for modelling of water for industrial use. This requires that the industrial demand is prescribed as an input to the model (see *prescribed_data.nml*).

TRUE
Consider demand for water for industrial use.

FALSE
Do not consider industrial demand.

`JULES_WATER_RESOURCES::l_water_irrigation`

Type
logical

Default
F

Switch for modelling of water for irrigation.

TRUE
Consider demand for water for irrigation. This must be used with `l_irrig_dmd` = TRUE (to activate the inclusion of irrigation in other aspects of the model) and `l_irrig_limit` = FALSE (to avoid triggering an alternative approach to calculating the water available for irrigation).

FALSE
Do not consider irrigation demand.

`JULES_WATER_RESOURCES::`**`l_water_livestock`**

Type
logical

Default
F

Switch for modelling of water for livestock.

TRUE
Consider demand for water for livestock. This requires that the livestock demand is prescribed as an input to the model (see *prescribed_data.nml*).

FALSE
Do not consider livestock demand.

`JULES_WATER_RESOURCES::`**`l_water_transfers`**

Type
logical

Default
F

Switch for modelling of water for transfers.

TRUE
Consider demand for water for transfers. This requires that the demand for transfers is prescribed as an input to the model (see *prescribed_data.nml*).

FALSE
Do not consider transfers.

Switches that control prioritisation of demands.

`JULES_WATER_RESOURCES::`**`l_prioritise`**

Type
logical

Default
F

Switch controlling prioritisation of demands.

TRUE
Rank demands from sectors in priority order.

FALSE
No prioritisation. No further items from this group are required.

`JULES_WATER_RESOURCES::`**`priority`**

Type
character

Default
none

A list of water sector names, in order of decreasing priority - see the table below for valid names. Only used if *l_prioritise* = TRUE. All active sectors (as selected by switches such as *l_water_domestic*) must be represented in this list. The same prioritisation is used for all points in the domain.

Name	Description
`dom`	Domestic use
`env`	Environmental flows
`ind`	Industrial use
`irr`	Irrigation
`liv`	Livestock use
`tra`	Water transfers

JULES_WATER_RESOURCES::**nr_gwater_model**

Type
integer

Permitted
0,1,2

Default
none

Choice for the model of non-renewable groundwater. Non-renewable groundwater is water that is not otherwise explicitly included in the model. It is an idealised, infinite source of water which is typically intended to allow consideration of pumping of grounwater from deep reserves that are difficult to quantify.

Possible values are:

0. No non-renewable groundwater is considered.
1. Non-renewable groundwater is used as a last resort, when no other sources of water are available.
2. Non-renewable groundwater is used as as 'part of the mix', in conjunction with other sources of water.

JULES_WATER_RESOURCES::**rf_domestic**

Type
real

Permitted
0-1

Default
none

The fraction of water that is returned after abstraction for domestic use (via sewage systems etc.). Only used if *l_water_domestic* = TRUE.

JULES_WATER_RESOURCES::**rf_industry**

Type
real

Permitted
0-1

Default
none

The fraction of water that is returned after abstraction for industrial use. Only used if *l_water_industry* = TRUE.

`JULES_WATER_RESOURCES::`**`rf_livestock`**

Type
real

Permitted
0-1

Default
none

The fraction of water that is returned after abstraction for livestock. Only used if *`l_water_livestock`* = TRUE.

6.17 `jules_irrig.nml`

This file sets the irrigation options. It contains one namelist called *`JULES_IRRIG`*.

6.17.1 `JULES_IRRIG` namelist members

This namelist specifies the different options available for setting up the irrigation.

Note:

Irrigation can be applied at a constant rate in three ways:

- 1. To apply a constant irrigation to all surface tiles the irrigation settings are as follows: frac_irrig_all_tiles=T, set_irrfrac_on_irrtiles=F and set a value for *`const_frac_irr`*.
- 2. To apply a constant irrigation to only specific surface tiles the irrigation settings are as follows: frac_irrig_all_tiles=F and set_irrfrac_on_irrtiles=T and set a value for *`const_irrfrac_irrtiles`*.
- 3. To apply a constant irrigation to specific surface tiles as an average across the gridbox, which is the way irrigation on specific tiles was done prior to vn5.7, the irrigation settings are as follows: frac_irrig_all_tiles=F, set_irrfrac_on_irrtiles=F and set a value for *`const_frac_irr`*.

In both option 2 and 3, *`irrigtiles`* is the index of surface tiles you wish to irrigate, the length of which is *`nirrtile`* e.g. if you include wheat and maize in your run at index 5 and 6 then irrigtiles=5,6 and nirrtile=2.

`JULES_IRRIG::`**`l_irrig_dmd`**

Type
logical

Default
F

Switch controlling the implementation of irrigation demand code.

TRUE
Tiles are irrigated.

FALSE
No effect.

This must be set to TRUE if *`l_water_irrigation`* = TRUE.

`JULES_IRRIG::`**`l_irrig_limit`**

Type
logical

Default
F

Switch controlling whether the amount of water used to irrigate tiles is limited.

TRUE
Water for irrigation is taken first from the deep soil (groundwater) store, and then from the river storage when the deep soil store is exhausted. Tiles are irrigated up to the critical point if the necessary water is available. This option requires *l_irrig_dmd* = TRUE, *l_top* = TRUE, *l_rivers* = TRUE and *i_river_vn* = `1,3`.

> **Warning:** The irrigation supply code in JULES is still in development, and is available in this release to support beta testing activities.
>
> Users should ensure that results are as expected, and provide feedback where deficiencies are identified.

FALSE
Tiles will be irrigated to critical point from an unconstrained water supply.

This must be set to FALSE if *l_water_irrigation* = TRUE.

`JULES_IRRIG::`**`irr_crop`**

Type
integer

Permitted
0, 1 or 2

Default
0

0. Irrigation season (i.e. season in which crops might be growing on the gridbox) lasts the entire year.
1. Irrigation season is determined from driving data according to *Döll & Siebert (2002)* method. No irrigation is applied outside the irrigation season.
2. Irrigation season is determined by maximum dvi across all surface tiles. Requires *ncpft* > 0. No irrigation is applied outside the irrigation season.

`JULES_IRRIG::`**`frac_irrig_all_tiles`**

Type
logical

Default
T

If T, then irrigation fraction is applied to all surface tiles, and F, it is applied only to the surface tiles specified in *irrigtiles*.

`JULES_IRRIG::`**`set_irrfrac_on_irrtiles`**

Type
logical

Default
F

If F then irrigation is applied as an average across the gridbox and not to specific surface tiles. If T, then the irrigation fraction is only applied to the surface tile specified in *irrigtiles*. Both *frac_irrig_all_tiles* and *set_irrfrac_on_irrtiles* cannot be set to T.

`JULES_IRRIG::`**`nirrtile`**

Type
integer

Default
None

The number of surface tiles to be irrigated. Only used if *frac_irrig_all_tiles* = F.

`JULES_IRRIG::`**`irrigtiles`**

Type
integer(nirrtile)

Default
None

Indices of the surface tiles to be irrigated. Only used if *frac_irrig_all_tiles* = F.

`JULES_IRRIG::`**`nstep_irrig`**

Type
integer

Permitted
> 0

Default
86400/*timestep_len*

The number of model timesteps per irrigation update step

Irrigation will be updated every *nstep_irrig* timesteps. For example, with a model timestep of 1 hour, *nstep_irrig* = 24 means that irrigation will be updated on the 24th timestep, i.e. daily updates.

nstep_irrig = NINT(frequency of irrigation update (in sec)) / *timestep_len*)

6.17.2 JULES_IRRIG references

- Döll, P., and Siebert, S., Global modeling of irrigation water requirements, Water Resour. Res., 38(4), https://doi.org/10.1029/2001WR000355, 2002.

6.18 science_fixes.nml

This file contains one namelist called *JULES_TEMP_FIXES*.

This namelist sets 'short-term' temporary logicals used to protect science bug fixes that lead to alterations in science results. It is expected that these logicals will be short lived as the preference should be for all configurations to use the corrected code. However, to maintain short term reproducibility of results across JULES versions the fixes are protected by logicals until the fixes become the default in all model configurations at which point the logical is retired. See module for when the switch is due for review.

6.18.1 JULES_TEMP_FIXES namelist members

`JULES_TEMP_FIXES::`**`ctile_orog_fix`**

Type
integer

Permitted
0-2

Default
2

If nonzero, corrects the surface exchange calculations in coastally tiled grid-boxes, assuming that the lowest level is physically terrain following and adjusting the temperature of the land/sea portions in accordance with their relative offset from the grid-box mean height using a dry/moist lapse rate where appropriate. Option 2 will only adjust values over the sea.

JULES_TEMP_FIXES::**l_accurate_rho**

Type
logical

Default
F

This switch improves the calculation of surface air density in the surface turbulent fluxes. It includes appropriate use of dry air density when the atmospheric water vapour is expressed as a mixing ratio (l_mr_physics = .TRUE.), otherwise use the wet air density when it is expressed as a specific humidity.

JULES_TEMP_FIXES::**l_dtcanfix**

Type
logical

Default
F

This switch corrects a bug in the evolution of the skin temperature in the implicit solver, whereby the change in the skin temperature is artificially constrained. This generally has a small effect, but can cause unphysical results if a canopy with a large heat capacity is coupled to an underlying substrate with a small heat capacity.

JULES_TEMP_FIXES::**l_fix_alb_ice_thick**

Type
logical

Default
F

When zero-layer sea ice is used the thermodynamics is calculated in the UM through an effective thickness calculated from snow and ice thicknesses and associated thermal conductivities. With multi-layer sea ice the thermodynamics is calculated in the sea ice component of the model, and the effective thickness is no longer required. However, it was still being used erroneously. This fix removes the effective thickness adjustment when multi-layer sea ice is used.

JULES_TEMP_FIXES::**l_fix_albsnow_ts**

Type
logical

Default
F

The original version of the two-stream scheme to calculate the albedo of snow in JULES contains a bug in the calculation of the reflection coefficient that renders very thin layers of snow too reflective. This logical applies the appropriate correction when it is enabled.

JULES_TEMP_FIXES::**l_fix_lake_ice_temperatures**

Type
logical

Default
F

If true, allows sea ice temperatures in lakes to evolve over time for coupled models when the lake is defined as a sea point but is not coupled to an ocean model.

JULES_TEMP_FIXES::**l_fix_moruses_roof_rad_coupling**

Type
logical

Default
F

If true, this switch corrects a bug in the surface energy balance when the MORUSES radiative roof coupling is used (see *l_moruses_storage*). If false, the thermal conductivity of the soil (hcons) is erroneously set to zero, which causes the roof to be effectively uncoupled when *l_vegcan_soilfx*.

JULES_TEMP_FIXES::**l_fix_osa_chloro**

Type
logical

Default
F

When set to false, the chlorophyll content used to determine the optical properties of water, for the ocean surface albedo, are specified in gm-3 when the parameterisation they use is defined in mg m-3. It is a short term logical until the code becomes the new default.

JULES_TEMP_FIXES::**l_fix_snow_frac**

Type
logical

Default
F

When set to false, there is the potential to have small snow mass, but a zero snow fraction due to machine precision in the calculations. This prevents sublimation or snow melt from removing the remaining snow mass, hence small values can persist. In addition to this there is a conceptual bug in the calculation of the fraction of potential evaporation because it does not add in canopy evaporation when the snow fraction is less than one. When set to true these issues are corrected and in addition the radiation calculations for snow fraction are also made consistent.

JULES_TEMP_FIXES::**l_fix_ustar_dust**

Type
logical

Default
F

If true, corrects how ustar is calculated in the exchange coefficient for dust deposition

JULES_TEMP_FIXES::**l_fix_wind_snow**

Type
logical

Default
F

If true, ensures that wind speed is calculated for use in snow unloading. If false, the wind speed for unloading will be zero on timesteps when 10m wind diagnostics are not calculated. This will tend to leave more snow on the vegetation. It is a short term logical until the code becomes the new default.

6.19 timesteps.nml

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

Warning: It is recommended that all times use not local time, but Coordinated Universal Time (UTC; known in some countries as Greenwich Mean Time GMT). The correct specification of the time is essential if the options causing the surface albedo to depend on the solar zenith angle are set. If the data are provided in local solar time, *l_local_solar_time* should be set to TRUE to assume local solar time throughout the model.

6.19.1 `JULES_TIME` namelist members

`JULES_TIME::l_360`

Type
logical

Default
F

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.

`JULES_TIME::l_leap`

Type
logical

Default
T

Switch indicating whether the calendar has leap years. This flag is not used if l_360=T.

TRUE
Leap years are modelled i.e. each year consists of 365 or 366 days.

FALSE
Each year consists of 365 days.

`JULES_TIME::l_local_solar_time`

Type
logical

Default
F

Switch indicating whether the time-stamping of the driving data and throughout the run is to be interpreted as local solar time, not UTC.

TRUE
The driving data and all times within the model are interpreted as being in local solar time, irrespective of any data attributes.

FALSE
The time convention applying within the model and the driving data is assumed to be UTC.

`JULES_TIME::timestep_len`

Type
integer

Permitted
>= 1

Default
None

Model timestep length in seconds (n.b. ‘special periods’ -1 (monthly) and -2 (annual) may not be used).

Typically, 30 or 60 minutes is chosen, depending on the driving data available.

> **Warning:** If the timestep is too long, the model becomes numerically unstable.

`JULES_TIME::`**`main_run_start`**

`JULES_TIME::`**`main_run_end`**

Type
character

Default
None

The start and end times for the integration.

Each run of JULES consists of an optional spin-up period and the ‘main run’ that follows the spin-up. See below for more about the specification of the spin-up. These variables specify the start and end times for the ‘main run’.

The times must be given in the format:

```
"yyyy-mm-dd hh:mm:ss"
```

`JULES_TIME::`**`print_step`**

Type
integer

Permitted
>= 1

Default
1

Number of timesteps between printing timestep information to screen, i.e. if *`print_step`* = 48, then the timestep start time will only be printed every 48 timesteps.

6.19.2 `JULES_SPINUP` namelist members

`JULES_SPINUP::`**`max_spinup_cycles`**

Type
integer

Permitted
>= 0

Default
0

The maximum number of times the spin-up period is to be repeated:

0
No spin-up.

> 0
At least 1 and at most *`max_spinup_cycles`* repetitions of spin-up are used.

After each repetition, the model tests whether the selected variables have changed by more than a specified amount over the last repetition (see *`tolerance`* below).

If the change is less than this amount, the model is considered to have spun up and the model moves on to the main run.

`JULES_SPINUP::`**`spinup_start`**

`JULES_SPINUP::`**`spinup_end`**

Type
character

Default
None

Only used if *max_spinup_cycles* > 0.

The start and end times for each cycle of spin-up.

The times must be given in the format:

```
"yyyy-mm-dd hh:mm:ss"
```

JULES_SPINUP::**terminate_on_spinup_fail**

Type
logical

Default
F

Only used if *max_spinup_cycles* > 0.

Switch controlling behaviour if the model does not pass the spin-up test after *max_spinup_cycles* of spin-up.

TRUE
End the run if model has not spun up.

FALSE
Continue the run regardless.

Variables used to specify spin-up conditions

JULES_SPINUP::**nvars**

Type
integer

Permitted
>= 0

Default
0

Only used if *max_spinup_cycles* > 0.

The number of variables to use to assess if the model has spun up.

JULES_SPINUP::**var**

Type
character(nvars)

Default
None

Only used if *nvars* > 0.

List of variables to be used to determine if the model has spun up. Spin-up can be assessed in terms of soil temperature and soil moisture.

Possible values are:

c_soil
Soil carbon in each layer (summed over all pools) (kg m^{-2}).

c_veg
Vegetation carbon (summed over all vegetation types) (kg m^{-2}).

smcl
Moisture content of each soil layer (kg m^{-2}).

t_soil
Temperature of each soil layer (K).

JULES_SPINUP::**use_percent**

Type
logical(nvars)

Default
F

Only used if *nvars* > 0.

Indicates whether the tolerance for each variable is expressed as a percentage.

TRUE
Tolerance is a percentage.

FALSE
Tolerance is an absolute value.

JULES_SPINUP::**tolerance**

Type
real(nvars)

Default
None

Only used if *nvars* > 0.

Tolerance for spin-up test for each variable.

For each spin-up variable, this is the maximum allowed change over a spin-up cycle if the variable is to be considered as spun-up.

6.19.3 Note on time conventions

When specifying start times (e.g. *main_run_start*, *spinup_start*), the time is taken to be the *start of the first timestep*. When specifying end times (e.g. *main_run_end*, *spinup_end*), the time is taken to be the *end of the last timestep*. Also, the time “00:00:00” always refers to midnight at the *start* of the day concerned. Take the following setup:

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

  # ...
/
```

With this setup, exactly one whole year of timesteps will be run. The first model timestep begins at 1997-01-01 00:00:00, the second at 1997-01-01 01:00:00 etc. The final model timestep begins at 1997-12-31 23:00:00 and ends at 1998-01-01 00:00:00. Note that even though only 1997 is simulated, JULES may nevertheless need to access data file(s) for subsequent timesteps (January 1998 in this example), depending on the interpolation flag settings in *Temporal interpolation*.

For example, if your driving data set extends to the end of 1997 only, then you may need to either stop the simulation 1-2 data timesteps before the end of 1997 (by modifying *main_run_end* and *spinup_end*; n.b. if these data timesteps are at the end of the year/month then an annual/monthly average will not be calculated) or generate dummy driving data for the start of 1998 (which must be realistic because they will be used for interpolation during the last few model timesteps).

The maximum extent of your driving data should be specified by `data_start` and `data_end`. The periods of the main run (`main_run_start` to `main_run_end`) and spin-up (`spinup_start` to `spinup_end`) must be contained within `data_start` to `data_end`.

Note that all times are recommended to be in Coordinated Universal Time (UTC), not local time (see Warning at top of this page). Note also the limitations on timestep mentioned in *Temporal interpolation* and on run length mentioned in *Known limitations of the code*.

6.19.4 Note on solar zenith angle

When the characteristics of the surface relevant to solar radiation are represented in a simple manner, the cosine of the solar zenith angle itself is not required and all that is needed is the downward shortwave flux provided by the forcing data. In such cases it is sufficient to set `l_cosz` = FALSE, which will set the cosine of the solar zenith angle to a default value of 1.0.

However, more elaborate treatments of the surface albedo and of solar radiative transfer in plant canopies do depend on the actual value of the cosine of the solar zenith angle, as well as the downward flux, so it is more realistic to set `l_cosz` = TRUE, and, indeed, the results obtained with `l_cosz` = FALSE may be significantly in error. To calculate the cosine of the zenith angle, the times of the forcing data must be specified accurately. The consistent use of UTC is strongly recommended. Nevertheless, certain forcing data, widely used within the land-surface community, are specified in local solar time, even though the metadata in the file may refer to UTC. In such cases `l_local_solar_time` should be set to TRUE when `l_cosz` = TRUE.

6.19.5 Examples

A run without spin-up

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

&JULES_SPINUP
  max_spinup_cycles = 0
/
```

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

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

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

&JULES_SPINUP
  max_spinup_cycles = 5,
  spinup_start      = "1996-01-01 00:00:00",
  spinup_end        = "1997-01-01 00:00:00"

  # <spinup variable specification>
/
```

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

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

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

&JULES_SPINUP
  max_spinup_cycles = 5,
  spinup_start      = "1997-01-01 00:00:00",
  spinup_end        = "1998-01-01 00:00:00"

  # <spinup variable specification>
/
```

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

Example of specifying requirements for spin-up

```
&JULES_SPINUP
  # <spinup time specification>

  terminate_on_spinup_fail = T,

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

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

6.19.6 Notes on spin-up

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

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

During the spin-up phase of a run, JULES provides the correct driving data (for example, meteorological data) as the model time 'cycles' round over the spin-up period. Consider the case of a spin-up from 2005-01-01 00:00:00

to 2006-01-01 00:00:00. At or near the end of 31st December 2005 during the spin-up, the driving data will start to adjust to the values for 1st January 2005. The calculated driving data may vary slightly between the start or end of the first cycle and similar times in later cycles, because of the need to match the data at the end of each cycle to that at the start of the next cycle. When the main run begins after a period of spin-up, the driving data is reset to the start of the main run - no effort is made to adjust the data for a smooth transition. Generally this does not cause a problem.

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

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

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

6.20 `model_grid.nml`

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

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

As discussed in *General principles*, the input grid consists of three pieces of information:

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

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

In most cases, the model grid will be represented internally as a vector of points, even when the input grid is 2D. Numerically, this makes no difference. The only time that the model grid will be 2D is when the input grid is 2D, `force_1d_grid` = F and the model grid is a contiguous rectangular subsection of the input grid.

6.20.1 `JULES_INPUT_GRID` namelist members

Warning: The dimension names specified in this namelist will be used for all input files.

JULES_INPUT_GRID::**grid_is_1d**

Type
logical

Default
F

Indicates if the input grid is 1D or 2D.

TRUE
Variables have one grid dimension in the input file(s) - a points dimensions (e.g. a vector of land points with grid dimension "land").

FALSE
Variables have two grid dimensions in the input file(s) - an x and a y dimension.

Only used when `grid_is_1d` = TRUE

JULES_INPUT_GRID::**grid_dim_name**

Type
character

Default
"land"

The name of the single grid dimension.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**npoints**

Type
integer

Permitted
>= 1

Default
0

The size of the single grid dimension.

Only used when `grid_is_1d` = FALSE

JULES_INPUT_GRID::**x_dim_name**

Type
character

Default
“x”

The name of the x dimension (it may, but does not have to, coincide with *x_dim_name*).

Note: For ASCII files, this can be anything. For NetCDF files, it should be the name of the dimension in the input file(s).

`JULES_INPUT_GRID::`**`y_dim_name`**

Type
character

Default
“y”

The name of the y dimension (it may, but does not have to, coincide with *y_dim_name*).

Note: For ASCII files, this can be anything. For NetCDF files, it should be the name of the dimension in the input file(s).

`JULES_INPUT_GRID::`**`nx`**

Type
integer

Permitted
>= 1

Default
0

The size of the x dimension.

`JULES_INPUT_GRID::`**`ny`**

Type
integer

Permitted
>= 1

Default
0

The size of the y dimension.

`JULES_INPUT_GRID::`**`time_dim_name`**

Type
character

Default
“time”

The name of the time dimension in any input files containing time varying data.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

`JULES_INPUT_GRID::`**`pft_dim_name`**

Type
character

Default
"pft"

The dimension name used when variables have an additional dimension of size *npft*.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**cpft_dim_name**

Type
character

Default
"cpft"

The dimension name used when variables have an additional dimension of size *ncpft*.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**nvg_dim_name**

Type
character

Default
"nvg"

The dimension name used when variables have an additional dimension of size *nnvg*.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**type_dim_name**

Type
character

Default
"type"

The dimension name used when variables have an additional dimension of size `ntype`.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**tile_dim_name**

Type
character

Default
"tile"

The dimension name used when variables have an additional dimension of size `nsurft`.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**soil_dim_name**

Type
character

Default
"soil"

The dimension name used when variables have an additional dimension of size *sm_levels*.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**snow_dim_name**

Type
character

Default
"snow"

The dimension name used when variables have an additional dimension of size *nsmax*.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**sclayer_dim_name**

Type
character

Default
"sclayer"

The dimension name used for the soil biogeochemistry when layered soil is used i.e. *l_layeredc* = TRUE. When *l_layeredc* = TRUE the soil biogeochemistry has the same number of layers as the soil hydrology (*sm_levels*). When *l_layeredc* = FALSE the soil biogeochemistry represents a single bulk layer. Despite the similar name, this parameter is unrelated to *dim_cslayer*.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**scpool_dim_name**

Type
character

Default
"scpool"

The dimension name used when variables have an additional dimension of size `dim_cs1`.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**bedrock_dim_name**

Type
character

Default
"bedrock"

The dimension name used when variables have an additional dimension of size *ns_deep*.

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**tracer_dim_name**

Type
character

Default
"tracer"

The dimension name used when variables have an additional dimension of size *ndry_dep_species* (e.g. chemical tracers in the atmosphere).

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

JULES_INPUT_GRID::**bl_level_dim_name**

Type
character

Default
"bllevel"

The dimension name used when variables have an additional dimension of size *bl_levels* (e.g. variables on atmospheric boundary layer levels).

Note: For ASCII files, this can be anything. For NetCDF files, it should the name of the dimension in input file(s).

6.20.2 JULES_LATLON namelist members

Members used to determine how gridpoint location variables are set

JULES_LATLON::**read_from_dump**

Type
logical

Default
F

TRUE
Populate variables associated with this namelist from the dump file. All other namelist members are ignored.

FALSE
Use the other namelist members to determine how to populate variables.

JULES_LATLON::**l_coord_latlon**

Type
logical

Default
F

TRUE
The coordinate system used for the model grid is latitude and longitude.

FALSE
The model grid is defined by projection coordinates other than latitude and longitude (e.g. northings and eastings, or a rotated grid).

`JULES_LATLON::`**`nvars`**

Type
integer

Permitted
>= 2

Default
0

The number of location variables that will be provided (see *List of grid location properties*).

`JULES_LATLON::`**`var`**

Type
character(nvars)

Default
None

List of location variable names as recognised by JULES (see *List of grid location properties*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_LATLON::`**`use_file`**

Type
logical(nvars)

Default
T

For each JULES variable specified in `var`, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using `const_val` below.

`JULES_LATLON::`**`var_name`**

Type
character(nvars)

Default
'' (empty string)

For each JULES variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from `var` is used instead.

This is not used for variables where `use_file` = FALSE, but a placeholder must still be given in that case.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

JULES_LATLON::**tpl_name**

Type
character(nvars)

Default
None

For each JULES variable specified in *var*, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

JULES_LATLON::**const_val**

Type
real(nvars)

Default
None

For each JULES variable specified in *var* where *use_file* = FALSE, this is a constant value that the variable will be set to at every point.

This is not used for variables where *use_file* = TRUE, but a placeholder must still be given in that case.

JULES_LATLON::**file**

Type
character

Default
None

The file to read ancillary properties from.

If *use_file* is FALSE for every variable, this will not be used.

This file name can use *variable name templating*.

List of grid location properties

The following table summarises ancillary fields that give the location and related characteristics of each point on the grid, specified from an ancillary file if *use_file* = TRUE.

Name	Description
latitude	Latitude of each point. Always required.
longitude	Longitude of each point. Always required. Values in the range -180 to 360 are allowed.
projection_x_coord	Values of the projection coordinate in the x direction. This is only required if *l_coord_latlon* = FALSE. Note that these can have any valid unit.
projection_y_coord	Values of the projection coordinate in the y direction. This is only required if *l_coord_latlon* = FALSE. Note that these can have any valid unit.
grid_area	The area of each gridbox (m:sup`2`) This is only requred if irrigation is being modelled with *l_water_resources* = TRUE and *l_water_irrigation* = TRUE.

Examples of how to specify the model domain using through this namelist are provided at the end of this section.

6.20.3 `JULES_LAND_FRAC` namelist members

Land fraction is the fraction of each gridbox that is land. Currently, JULES considers any gridbox with land fraction > 0 to be 100% land, and all others to be 100% sea (or sea-ice). Land fraction data can be used to select only land points from the full input grid (see below).

Warning: When the input grid consists of a single location (1D and *npoints* = 1 or 2D and *nx* = *ny* = 1), that single location is assumed to be 100% land.

For any input grid with more than a single location, the following are used:

`JULES_LAND_FRAC::`**`file`**

Type
character

Default
None

The name of the file to read land fraction data from.

`JULES_LAND_FRAC::`**`land_frac_name`**

Type
character

Default
'land_fraction'

The name of the variable containing the land fraction data.

In the file, the variable must have no levels dimensions and no time dimension.

6.20.4 `JULES_MODEL_GRID` namelist members

Members of this namelist are used to select the points to be modelled from the input grid. This can be done in various ways (see the *Examples of grid setups*).

`JULES_MODEL_GRID::`**`land_only`**

Type
logical

Default
T

TRUE
Model land points only (from the points that are selected with other options).

FALSE
Model all selected points.

If *use_subgrid* = FALSE (see below), the land points will be extracted from the full input grid.

If *use_subgrid* = TRUE, then the subgrid extraction takes place first, and the land points will be extracted from the specified subgrid.

`JULES_MODEL_GRID::`**`force_1d_grid`**

Type
logical

Default
F

TRUE
Force the model grid to be 1D, even if it would otherwise have been 2D.

FALSE
The model grid takes its default shape.

`JULES_MODEL_GRID::`**`use_subgrid`**

Type
logical

Default
F

TRUE
The model grid is a subset of the full input grid, specified using some valid combination of the options below.

FALSE
The model grid is the full input grid.

Only used if `use_subgrid` = TRUE

`JULES_MODEL_GRID::`**`l_bounds`**

Type
logical

Default
None

TRUE
Subset of points to model will be selected using bounds for the coordinates variables.

FALSE
Subset of points to model will be selected using a list of coordinate pairs for each point.

If *`l_coord_latlon`* = TRUE, the coordinates used here are latitude and longitude.

If *`l_coord_latlon`* = FALSE, the coordinates used here are the values stored in the variables projection_x_coord and projection_y_coord.

Only used if `l_bounds` = TRUE

`JULES_MODEL_GRID::`**`y_bounds`**

Type
real(2)

Default
None

The lower and upper bounds (in that order) for the y coordinate used to select points. Assuming that the coordinate is latitude (see *`l_coord_latlon`*) the model grid will comprise the points where `y_bounds(1) <= latitude <= y_bounds(2)`.

`JULES_MODEL_GRID::`**`x_bounds`**

Type
real(2)

Default
None

The lower and upper bounds (in that order) for the x coordinate used to select points. Assuming that the coordinate is longitude (see *l_coord_latlon*) the model grid will comprise the points where `x_bounds(1) <= longitude <= x_bounds(2)`.

If the x coordinate is longitude, the values of x_bounds should lie in the range [-180, 360]. A special case is that in which the desired subgrid straddles the edge of a global input grid. For example, if the input grid has longitudes in [0, 360] and a domain of 20 degrees of longitude centred on 0degE is required, this should be indicated using `xbounds=-10,10` (not xbounds=360,370 because values > 360 are not recognised). In this case the JULES code recognises the cyclic nature of longitude and correctly picks up points in both hemispheres, even though -10degE is outwith the longitude values in the input grid.

Only used if `l_bounds` = FALSE

JULES_MODEL_GRID::**npoints**

Type
integer

Permitted
>= 1

Default
0

The number of points to model.

JULES_MODEL_GRID::**points_file**

Type
character

Default
None

The name of the file containing the coordinates for each point.

If *l_coord_latlon* = TRUE, the coordinates used here are latitude and longitude. Each line in the file should contain the latitude and longitude (in that order) for a point.

If *l_coord_latlon* = FALSE, the coordinates used here are the values stored in the variables projection_x_coord and projection_y_coord. Each line in the file should contain the value for projection_y_coord and projection_x_coord (in that order; note this is y then x) for a point.

An error is raised and the run terminates if any coordinate pair does not match to a location in the input grid.

6.20.5 JULES_NLSIZES namelist members

This namelist is used to set the number of levels in the boundary layer.

JULES_NLSIZES::**bl_levels**

Type
integer

Default
1

Number of boundary layer levels. This is only used if atmospheric deposition is selected (*l_deposition* = TRUE) in which case it is used to set the size of input fields.

6.20.6 `JULES_SURF_HGT` namelist members

This namelist sets the elevation of each surface tile. Elevations can either be *relative to the gridbox mean* or *have constant elevation bands above sea-level*.

If tile elevations are set relative to the gridbox mean, then the gridbox mean elevation is not required. The gridbox mean elevation is implicit in the near-surface meteorological data that are provided (higher locations will tend to be colder, have lower pressure, etc.). The elevation of each tile is used to alter the values of the air temperature and humidity (and possibly downwelling longwave, see *`l_elev_lw_down`*) over that tile relative to the surface.

If any tile uses absolute heights (i.e. *`l_elev_absolute_height`* has at least one element that is .true.), then the gridbox mean elevation must also be supplied. This is read in from the optional *`JULES_Z_LAND`* namelist which is described below. The model calculates elevations relative to the gridbox mean by taking the difference between the absolute elevation and the gridbox mean.

If any tile uses absolute heights, then tile heights are set constant across a domain, regardless of whether each tile's height is specified as a relative offset or absolute. This makes it simple to set zero-offset heights for tiles that should not be considered in the elevation bands. It is no longer possible to have spatially varying tile heights if this option is used.

`JULES_SURF_HGT::`**`zero_height`**

Type
logical

Default
T

Switch used to simplify the initialisation of tile elevation.

Note: If *`l_aggregate`* = TRUE, this switch is also set to TRUE.

TRUE
Set all surface tile elevations to zero. This is a very common configuration.

FALSE
Set surface tile heights using specified data.

Only used if `zero_height` = FALSE

`JULES_SURF_HGT::`**`l_elev_absolute_height`**

Type
logical(nsurft)

Default
F

TRUE
Heights of surface tiles are absolute values above sea-level. If this option is used, then the elevation of the forcing data must also be provided (see *`JULES_Z_LAND`* namelist below).

FALSE
Surface tile heights are relative to the gridbox mean.

`JULES_SURF_HGT::`**`use_file`**

Type
logical

Default
T

This indicates if surface tile heights relative to the gridbox mean should be read from a specified file or namelist.

TRUE
The variable will be read from a file if the input grid consists of more than location.

FALSE
The variable will be read from a namelist if the input grid is for a single location.

Only used if `use_file` = TRUE

`JULES_SURF_HGT::`**`file`**

Type
character

Default
None

The name of the file containing surface tile heights relative to the gridbox mean.

`JULES_SURF_HGT::`**`surf_hgt_name`**

Type
character

Default
'surf_hgt'

The name of the variable containing surface tile heights relative to the gridbox mean. In the file, the variable must have a single levels dimension of size `nsurft` called *`tile_dim_name`*.

Only used if `use_file` = FALSE

`JULES_SURF_HGT::`**`surf_hgt_io`**

Type
real(nsurft)

Default
None

Surface tile heights relative to the gridbox mean for a single location.

6.20.7 `JULES_Z_LAND` namelist members

This is an optional namelist and only used if any surface tile has *`l_elev_absolute_height`* = TRUE. The namelist sets values for the elevation bands and reads the elevation of the forcing data.

`JULES_Z_LAND::`**`surf_hgt_band`**

Type
real(nsurft)

Default
None

Spatially invariant elevation bands for each surface tile. These may be relative to the gridbox mean or absolute elevations above sea-level depending on *`l_elev_absolute_height`*.

`JULES_Z_LAND::`**`use_file`**

Type
logical

Default
T

This indicates if the elevation of the forcing data should be read from a file or from a namelist.

TRUE
The variable will be read from a file if the input grid consists of more than location.

FALSE
The variable will be read from a namelist if the input grid is for a single location.

Used if `use_file` = TRUE

`JULES_Z_LAND::file`

Type
character

Default
None

The name of the file containing the elevation of the forcing data.

`JULES_Z_LAND::z_land_name`

Type
character

Default
'z_land'

The name of the variable containing the elevation of the forcing data. In the file, the variable must have no level dimensions and no time dimensions.

Used if `use_file` = FALSE

`JULES_Z_LAND::z_land_io`

Type
real

Default
None

Elevation of the forcing data for a single location.

Example

The following gives an example of how you would set up the namelists to use elevation bands above sea-level.

```
&JULES_SURF_HGT

  zero_height            = .false.,

  # No elevation correction to surface tiles 1 to 6, use elevation bands for surface
↪tiles 7 to 9
  l_elev_absolute_height = 6*.false., 3*.true.,

/

&JULES_Z_LAND

```

(continues on next page)

(continued from previous page)

```
  # Set values for the elevation bands.
  surf_hgt_band          = 6*0.0, 1000.0, 2000.0, 3000.0,

  # Read the WFDEI forcing data elevation from a file
  use_file               = .true.,
  file                   = 'WFDEI-elevation.nc',
  z_land_name            = 'elevation'

/
```

6.20.8 Examples of grid setups

A single location

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

&JULES_LATLON
  l_coord_latlon = T
  nvars     = 2,
  var       = 'latitude','longitude',
  use_file  = .false., .false.,
  const_val = 52.168, 5.744
/

&JULES_LAND_FRAC /

&JULES_MODEL_GRID /

&JULES_SURF_HGT
  zero_height = T
/
```

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

JULES_LATLON
The latitude and longitude of the single location are specified directly in the namelist. *nvars* = 2 indicates that the two mandatory variables will be provided, and *var* = 'latitude','longitude' confirms that these are the latitude and longitude. *use_file* = .false. indicates that the values will be read from the namelist (not from another file) and the values are provided after *const_val*.

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

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

Examples of gridded runs

All the examples in this section assume gridded NetCDF data.

Specifying a 1D input grid

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

```
&JULES_INPUT_GRID
  grid_is_1D = T,

  npoints = 15238,
  grid_dim_name = "land",

  time_dim_name = "time"
/
```

Specifying a 2D input grid

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

```
&JULES_INPUT_GRID
  grid_is_1D = F,

  nx = 96,
  ny = 56,

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

  tile_dim_name = "pseudo",

  time_dim_name = "time"
/
```

Specifying a subgrid using a given range of latitude and longitude

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

```
&JULES_LATLON
  l_coord_latlon = T,
  nvars       = 2,
  var         = 'latitude','longitude',
  use_file    = .true., .true.,
  file        = 'lat_lon.nc',
/

&JULES_LAND_FRAC
  file = 'land_mask.nc',
```

(continues on next page)

(continued from previous page)

```
  land_frac_name = 'land_frac'
/

&JULES_MODEL_GRID
  land_only = F,

  use_subgrid = T,

  l_bounds = T,

  y_bounds = 55.0  57.0,
  x_bounds = -5.0  -3.0
/
```

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

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

Specifying a subgrid using a given range of projection coordinates (not latitude and longitude)

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

```
&JULES_LATLON
  l_coord_latlon = F,
  nvars      = 4,
  var        = 'latitude','longitude','projection_x_coord','projection_y_coord'
  use_file   = .true., .true., .true., .true.,
  file       = 'lat_lon.nc',
/

&JULES_LAND_FRAC
  file = 'land_mask.nc',

  land_frac_name = 'land_frac'
/

&JULES_MODEL_GRID
  land_only = F,

  use_subgrid = T,

  l_bounds = T,

  y_bounds = 500.0  40500.0,
  x_bounds = 25500.0 55500.0
/
```

In this setup *l_coord_latlon* = FALSE indicates that data will be read from a grid that is not defined by latitude and longitude - rather it uses other projection coordinates such as the northings and eastings of the Ordnance Survey (British) National Grid (BNG) OSGB36. The projection coordinates are read via the variables projection_x_coord and projection_y_coord. Note that the latitude and longitude of each point is also read in; JULES includes these in output files for reference, and they can also be required by the science code (e.g. for solar zenith angle).

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

Specifying a subgrid using a list of points

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

```
&JULES_LATLON
  l_coord_latlon = T,
  nvars      = 2,
  var        = 'latitude','longitude',
  use_file   = .true., .true.,
  file       = 'lat_lon.nc',
/

&JULES_LAND_FRAC
  file = 'land_mask.nc',

  land_frac_name = 'land_frac'
/

&JULES_MODEL_GRID
  use_subgrid = T,

  l_bounds = F,

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

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

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

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

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

The only configuration that yields a 2D model grid

```
&JULES_INPUT_GRID
  grid_is_1d = F,

  nx = 96,
  ny = 56,

  # ...
/

&JULES_LATLON
  # <specified from file>
/

&JULES_LAND_FRAC
  # <specified from file>
/

&JULES_MODEL_GRID
  land_only = F
/
```

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

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

If the input grid is a 2D region where every point is land (i.e. not the whole globe), then *land_only* = TRUE would also yield a 2D model grid. If any options are set that mean some points from the input grid are not modeled, the model grid will be a vector of points. Computationally, this makes no difference.

6.21 ancillaries.nml

This file sets up spatially varying ancillary values. It contains the following namelists: *JULES_FRAC*, *JULES_VEGETATION_PROPS*, *JULES_SOIL_PROPS*, *JULES_TOP*, *JULES_PDM*, *JULES_AGRIC*, *JULES_CROP_PROPS*, *JULES_IRRIG_PROPS*, *JULES_RIVERS_PROPS*, *JULES_OVERBANK_PROPS*, *JULES_WATER_RESOURCES_PROPS*, *URBAN_PROPERTIES* , *JULES_CO2* and *JULES_FLAKE*.

Data associated with each of these namelists can optionally be read from the dump file (if present) by setting `read_from_dump` to true. This functionality provides closer alignment with UM functionality and can help ensure that the correct ancillary data remain associated with the model state.

6.21.1 JULES_FRAC namelist members

This namelist specifies the fraction of the land area in each gridbox that is covered by each of the surface types. If *l_veg_compete* = TRUE, then the fraction of each surface type is modelled and the initial state should be specified in *JULES_INITIAL*. In all other cases, it must be read here.

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

If using either URBAN-2T or MORUSES then the total *urban* fraction can be specified instead of the separate *urban_canyon* and *urban_roof* contributions. When initialising, if the roof fraction is zero, the canyon fraction will be interpreted as the total *urban* fraction and be partitioned according to canyon fraction (W/R, see *URBAN_PROPERTIES*).

Note: For runs with dynamic vegetation (TRIFFID, *l_triffid* = TRUE) and *l_veg_compete* = TRUE, then the fraction of each surface type is being modelled and the initial state should be specified in *JULES_INITIAL* (which will override any settings given in this section). In all other cases, frac must be read here.

`JULES_FRAC::`**`read_from_dump`**

Type
logical

Default
F

TRUE
Populate variables associated with this namelist from the dump file. All other namelist members are ignored.

FALSE
Use the other namelist members to determine how to populate variables.

`JULES_FRAC::`**`file`**

Type
character

Default
None

The name of the file to read surface type fractional coverage data from.

`JULES_FRAC::`**`frac_name`**

Type
character

Default
'frac'

The name of the variable containing the surface type fractional coverage data.

Note: This is only used for NetCDF files. For ASCII files, the surface type fractional coverage data is expected to be the first (ideally only) variable in the file.

Note: The open water fraction of this array (given by *lake*) should contain permanent water, and wetland extents that are not being otherwise simulated. If groundwater inundation is being simulated (i.e. TOPMODEL is active *l_top* = TRUE and therefore fsat is being calculated) then all groundwater-maintained wetlands must be excluded from *frac_name*. If overbank inundation is being simulated (i.e. *l_riv_overbank* = TRUE) then all fluvial wetlands must be excluded from *frac_name*. Finally, note that simulation of a potential future climate scenario with greatly reduced areas for lakes that are currently 'permanent' would require suitable modification of the ancillary provided here.

In the file, the variable must have a single levels dimension of size `ntype` called *type_dim_name*, and should not have a time dimension.

6.21.2 `JULES_VEGETATION_PROPS` namelist members

This namelist specifies how spatially-varying properties of vegetation should be set.

At present only one variable - `t_home_gb` - can be specified via this namelist, and this is only required if thermal adaptation of photosynthetic capacity is selected (*`photo_acclim_model`* = 1 or 3).

Note that Leaf Area Index and vegetation height are specified elsewhere - see *`JULES_PRESCRIBED`*.

`JULES_VEGETATION_PROPS::`**`read_from_dump`**

Type
logical

Default
F

TRUE
Populate variables associated with this namelist from the dump file. All other namelist members are ignored.

FALSE
Use the other namelist members to determine how to populate variables.

`JULES_VEGETATION_PROPS::`**`file`**

Type
character

Default
None

The file to read vegetation properties from.

If *`use_file`* is FALSE for every variable, this will not be used.

This file name can use *variable name templating*.

`JULES_VEGETATION_PROPS::`**`nvars`**

Type
integer

Permitted
>= 0

Default
0

The number of vegetation property variables that will be provided (see *List of vegetation parameters*).

`JULES_VEGETATION_PROPS::`**`var`**

Type
character(nvars)

Default
None

List of vegetation variable names as recognised by JULES (see *List of vegetation parameters*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_VEGETATION_PROPS::`**`use_file`**

Type
logical(nvars)

Default
T

For each JULES variable specified in *var*, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using *const_val* below.

JULES_VEGETATION_PROPS::**var_name**

Type
character(nvars)

Default
'' (empty string)

For each JULES variable specified in *var* where *use_file* = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from *var* is used instead.

This is not used for variables where *use_file* = FALSE, but a placeholder must still be given in that case.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

JULES_VEGETATION_PROPS::**tpl_name**

Type
character(nvars)

Default
None

For each JULES variable specified in *var*, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

JULES_VEGETATION_PROPS::**const_val**

Type
real(nvars)

Default
None

For each JULES variable specified in *var* where *use_file* = FALSE, this is a constant value that the variable will be set to at every point.

This is not used for variables where *use_file* = TRUE, but a placeholder must still be given in that case.

List of vegetation parameters

Name	Description
t_home_gb	Average temperature (home temperature) for thermal adaptation of photosynthetic capacity (K), e.g. a multi-decadal average or pre-industrial temperature. Suggestions as to how to calculate a suitable temperature can be found in *Kattge and Knorr (2007)* or *Kumarathunge et al (2019)*. This variable should not have a time dimension nor any "levels" dimension.

6.21.3 JULES_SOIL_PROPS namelist members

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

JULES_SOIL_PROPS::**read_from_dump**

Type
logical

Default
F

TRUE
Populate variables associated with this namelist from the dump file. All other namelist members are ignored.

FALSE
Use the other namelist members to determine how to populate variables.

JULES_SOIL_PROPS::**const_z**

Type
logical

Default
F

Switch indicating if soil properties are to be uniform with depth.

TRUE
Soil characteristics do not vary with depth.

FALSE
Soil characteristics vary with depth. For any variable this is ignored if a constant value is to be used (see `const_val`).

JULES_SOIL_PROPS::**file**

Type
character

Default
None

The file to read soil properties from.

If `use_file` is FALSE for every variable, this will not be used.

This file name can use *variable name templating*.

JULES_SOIL_PROPS::**nvars**

Type
integer

Permitted
>= 0

Default
0

The number of soil property variables that will be provided (see *List of soil parameters*).

`JULES_SOIL_PROPS::`**`var`**

Type
character(nvars)

Default
None

List of soil variable names as recognised by JULES (see *List of soil parameters*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_SOIL_PROPS::`**`use_file`**

Type
logical(nvars)

Default
T

For each JULES variable specified in `var`, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using `const_val` below.

`JULES_SOIL_PROPS::`**`var_name`**

Type
character(nvars)

Default
'' (empty string)

For each JULES variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from `var` is used instead.

This is not used for variables where `use_file` = FALSE, but a placeholder must still be given in that case.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

`JULES_SOIL_PROPS::`**`tpl_name`**

Type
character(nvars)

Default
None

For each JULES variable specified in `var`, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

`JULES_SOIL_PROPS::`**`const_val`**

Type
real(nvars)

Default
None

For each JULES variable specified in *var* where *use_file* = FALSE, this is a constant value that the variable will be set to at every point in every layer (overriding *const_z* = FALSE).

This is not used for variables where *use_file* = TRUE, but a placeholder must still be given in that case.

List of soil parameters

If *const_z* = FALSE, variables read from file must have a single levels dimension. For most variables this dimension must be of size *sm_levels* and called *soil_dim_name*; exceptions to this rule are indicated in the table below.

If *const_z* = TRUE, variables read from file must have no levels dimensions.

If soil tiling is selected (*l_tile_soil* = TRUE), ancillary fields can be specified for each soil tile (*l_broadcast_ancils* = FALSE), or values can be read for one soil tile and copied to all tiles (*l_broadcast_ancils* = TRUE).

In all cases, the variables must have no time dimension.

Name	Description	Levels name
`albsoil`	Soil albedo. A single (averaged) waveband is used.	None
`b`	**Exponent in soil hydraulic characteristics.** n.b. Related to the Brooks & Corey parameter lambda by b=1/lambda and to the van Genuchten-Mualem parameter n by b=1/(n-1)	*soil_dim_name*
`hcap`	Dry heat capacity (J m^{-3} K^{-1}).	*soil_dim_name*
`hcon`	Dry thermal conductivity (W m^{-1} K^{-1}).	*soil_dim_name*
`satcon`	Hydraulic conductivity at saturation (kg m^{-2} s^{-1}).	*soil_dim_name*
`sathh`	If *l_vg_soil* = TRUE (i.e. using van Genuchten model), `sathh = 1 / alpha`, where `alpha` (m^{-1}) is a parameter of the van Genuchten model. If *l_vg_soil* = FALSE (using Brooks and Corey model), `sathh` is the soil matric suction at saturation (in pressure head units, m), i.e. the absolute value of the soil matric potential at saturation.	*soil_dim_name*
`sm_crit`	Volumetric soil moisture content at the critical point (m^3 water per m^3 soil). If *l_use_pft_psi* = F, the point at which soil moisture stress starts to restrict transpiration is a function of `sm_crit`, `sm_sat` and the pft-dependent parameter *fsmc_p0_io* . `sm_crit` is also used to calculate the surface conductance of bare soil.	*soil_dim_name*
`sm_sat`	Volumetric soil moisture content at saturation (m^3 water per m^3 soil). **Note:** This field is used to distinguish between soil points and land ice points. `sm_sat > 0` indicates a soil point.	*soil_dim_name*
`sm_wilt`	Volumetric soil moisture content at the wilting point (m^3 water per m^3 soil). If *l_use_pft_psi* = F, `sm_wilt` is the limit where soil moisture stress completely prevents transpiration. `sm_wilt` is also used to calculate soil respiration.	*soil_dim_name*
`clay`	Soil clay content (g clay per g soil). Only required for the 4-pool and ECOSSE soil carbon models (*soil_bgc_model* = 2 or 3). **Note:** To allow backwards compatibility when using the 4-pool model (*soil_bgc_model* = 2), if the clay content is not available it is set to 0.0 in the code. However, this is wrong - if it is not available it should be set to 0.23.	*sclayer_dim_name*
`soil_ph`	Soil pH. Only required for the ECOSSE soil carbon model (*soil_bgc_model* = 3).	*sclayer_dim_name*

6.21.4 `JULES_TOP` namelist members

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

`JULES_TOP::`**`read_from_dump`**

> **Type**
> logical
>
> **Default**
> F

TRUE
Populate variables associated with this namelist from the dump file. All other namelist members are ignored.

FALSE
Use the other namelist members to determine how to populate variables.

`JULES_TOP::`**`file`**

> **Type**
> character
>
> **Default**
> None

The file to read TOPMODEL properties from.

If *`use_file`* is FALSE for every variable, this will not be used.

This file name can use *variable name templating*.

`JULES_TOP::`**`nvars`**

> **Type**
> integer
>
> **Permitted**
> >= 0
>
> **Default**
> 0

The number of TOPMODEL property variables that will be provided. At present, all variables are required for runs using TOPMODEL (see *List of TOPMODEL parameters*).

`JULES_TOP::`**`var`**

> **Type**
> character(nvars)
>
> **Default**
> None

List of TOPMODEL variable names as recognised by JULES (see *List of TOPMODEL parameters*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_TOP::`**`use_file`**

> **Type**
> logical(nvars)

Default
T

For each JULES variable specified in *var*, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using *const_val* below.

JULES_TOP::**var_name**

Type
character(nvars)

Default
‘’ (empty string)

For each JULES variable specified in *var* where *use_file* = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from *var* is used instead.

This is not used for variables where *use_file* = FALSE, but a placeholder must still be given in that case.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

JULES_TOP::**tpl_name**

Type
character(nvars)

Default
None

For each JULES variable specified in *var*, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

JULES_TOP::**const_val**

Type
real(nvars)

Default
None

For each JULES variable specified in *var* where *use_file* = FALSE, this is a constant value that the variable will be set to at every point in every layer.

This is not used for variables where *use_file* = TRUE, but a placeholder must still be given in that case.

List of TOPMODEL parameters

All of the TOPMODEL variables listed below are expected to have no levels dimensions and no time dimension.

Name	Description
`fexp`	Decay factor describing how the saturated hydraulic conductivity decreases with depth below the standard soil column (m^{-1}). Routinely set between 2 and 3 m^{-1}. Gedney & Cox (2003, J Hydromet) used value 0.5 m^{-1}; Niu & Yang (2003, Global & Planet. Change) suggested a global mean value of 2.0 m^{-1}.
`ti_mean`	(Spatial, not temporal) mean value of the topographic index in each gridbox. Value 5.99 is the global mean given in Marthews et al. (2015, HESS)
`ti_sig`	(Spatial, not temporal) standard deviation of the topographic index in each gridbox. Values <0.5 are updated to =0.5 internally to allow at least some variability

6.21.5 `JULES_PDM` namelist members

This namelist reads spatially varying parameter values for the PDM-type parameterisation of runoff. The values are only used if *`l_pdm`* = TRUE. The description below is very brief. For further details, see the references under *`l_pdm`*.

`JULES_PDM::`**`file`**

Type
character

Default
None

The file to read PDM properties from.

If *`use_file`* is FALSE for every variable, this will not be used.

This file name can use *variable name templating*.

`JULES_PDM::`**`nvars`**

Type
integer

Permitted
>= 0

Default
0

The number of PDM property variables that will be provided (see *List of PDM parameters*). At present, only the topographic slope can be provided.

`JULES_PDM::`**`var`**

Type
character(nvars)

Default
None

List of PDM variable names as recognised by JULES (see *List of PDM parameters*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_PDM::use_file`

Type
logical(nvars)

Default
T

For each JULES variable specified in *`var`*, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using *`const_val`* below.

`JULES_PDM::var_name`

Type
character(nvars)

Default
None

For each JULES variable specified in *`var`* where *`use_file`* = TRUE, this is the name of the variable in the file containing the data.

This is not used for variables where *`use_file`* = FALSE, but a placeholder must still be given.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

`JULES_PDM::tpl_name`

Type
character(nvars)

Default
None

For each JULES variable specified in *`var`*, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

`JULES_PDM::const_val`

Type
real(nvars)

Default
None

For each JULES variable specified in *`var`* where *`use_file`* = FALSE, this is a constant value that the variable will be set to at every point in every layer. make html This is not used for variables where *`use_file`* = TRUE, but a placeholder must still be given.

List of PDM parameters

All of the PDM variables listed below are expected to have no levels dimensions and no time dimension.

Name	Description
`slope`	Mean value of the topographic slope in the gridbox (deg).

6.21.6 `JULES_AGRIC` namelist members

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

`JULES_AGRIC::`**`read_from_dump`**

Type
logical

Default
F

TRUE
Populate frac_agr, frac_past, and frac_biocrop from the dump file. All other namelist members are ignored.

FALSE
Use the other namelist members to determine how to populate variables.

`JULES_AGRIC::`**`zero_agric`**

Type
logical

Default
T

Switch used to simplify the initialisation of agricultural fraction.

TRUE
Set agricultural fraction at all points to zero.

FALSE
Set agricultural fraction using specified data.

Used if `zero_agric` = FALSE and the input grid consists of a single location

`JULES_AGRIC::`**`frac_agr`**

Type
real

Default
None

The agricultural fraction for the single location.

Used if `zero_agric` = FALSE and the input grid consists of more than one location

`JULES_AGRIC::`**`file`**

Type
character

Default
None

The name of the file to read agricultural fraction data from.

`JULES_AGRIC::`**`agric_name`**

Type
character

Default
'frac_agr'

The name of the variable containing the agricultural fraction data.

In the file, the variable must have no levels dimensions and no time dimension.

`JULES_AGRIC::`**`zero_past`**

Type
logical

Default
T

Switch used to simplify the initialisation of pasture fraction. Pasture fraction can only be used if *`l_trif_crop`* is TRUE.

TRUE
Set pasture fraction at all points to zero.

FALSE
Set pasture fraction using specified data.

Used if `zero_past` = FALSE and the input grid consists of a single location

`JULES_AGRIC::`**`frac_past`**

Type
real

Default
None

The pasture fraction for the single location.

Used if `zero_past` = FALSE and the input grid consists of more than one location

`JULES_AGRIC::`**`file_past`**

Type
character

Default
None

The name of the file to read pasture fraction data from.

`JULES_AGRIC::`**`pasture_name`**

Type
character

Default
'frac_past'

The name of the variable containing the pasture fraction data.

In the file, the variable must have no levels dimensions and no time dimension.

JULES_AGRIC::**zero_biocrop**

Type
logical

Default
T

Switch used to simplify the initialisation of bioenergy fraction. Bioenergy fraction can only be used if *l_trif_biocrop* is TRUE.

TRUE
Set bioenergy fraction at all points to zero.

FALSE
Set bioenergy fraction using specified data.

Used if zero_biocrop = FALSE and the input grid consists of a single location

JULES_AGRIC::**frac_biocrop**

Type
real

Default
None

The bioenergy fraction for the single location.

Used if zero_biocrop = FALSE and the input grid consists of more than one location

JULES_AGRIC::**file_biocrop**

Type
character

Default
None

The name of the file to read bioenergy fraction data from.

JULES_AGRIC::**biocrop_name**

Type
character

Default
'frac_biocrop'

The name of the variable containing the bioenergy fraction data.

In the file, the variable must have no levels dimensions and no time dimension.

Specify the day of year on which harvesting occurs. Only used if l_trif_biocrop = TRUE. A placeholder value must be set for all PFTs, though will only be used for PFTs with harvest_type_io = 2.

JULES_AGRIC::**read_harvest_doy_from_dump**

Type
logical

Default
F

TRUE
Populate harvest_doy from the dump file. All other namelist members are ignored.

FALSE
Use the other namelist members to determine how to populate variables.

`JULES_AGRIC::`**`file_harvest_doy`**

Type
character

Default
None

The name of the file to read harvest day-of-year data from.

`JULES_AGRIC::`**`harvest_doy_name`**

Type
character

Default
'harvest_doy'

The name of the variable containing the harvest day-of-year data.

Note: This is only used for NetCDF files. For ASCII files, the harvest day-of-year data is expected to be the first (ideally only) variable in the file.

6.21.7 `JULES_CROP_PROPS` namelist members

`JULES_CROP_PROPS::`**`read_from_dump`**

Type
logical

Default
F

TRUE
Populate variables associated with this namelist from the dump file. All other namelist members are ignored.

FALSE
Use the other namelist members to determine how to populate variables.

`JULES_CROP_PROPS::`**`file`**

Type
character

Default
None

The file from which crop properties are read.

If *`use_file`* is FALSE for every variable, this will not be used.

This file name can use *variable name templating*.

`JULES_CROP_PROPS::`**`nvars`**

Type
integer

Permitted
>= 0

Default
0

The number of crop property variables that will be provided (see *List of spatially-varying crop properties*).

`JULES_CROP_PROPS::var`

Type
character(nvars)

Default
None

List of variable names for spatially-varying crop properties as recognised by JULES (see *List of spatially-varying crop properties*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_CROP_PROPS::use_file`

Type
logical(nvars)

Default
T

For each JULES variable specified in `var`, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using `const_val` below.

`JULES_CROP_PROPS::var_name`

Type
character(nvars)

Default
'' (empty string)

For each JULES variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from `var` is used instead.

This is not used for variables where `use_file` = FALSE, but a placeholder must still be given in that case.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

`JULES_CROP_PROPS::tpl_name`

Type
character(nvars)

Default
None

For each JULES variable specified in `var`, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

JULES_CROP_PROPS::**const_val**

Type
real(nvars)

Default
None

For each JULES variable specified in *var* where *use_file* = FALSE, this is a constant value that the variable will be set to at every point in every layer.

This is not used for variables where *use_file* = TRUE, but a placeholder must still be given in that case.

List of spatially-varying crop properties

All of the crop variables listed below are expected to have a single levels dimension of size *ncpft* called *cpft_dim_name*.

Name	Description
`cropsowdate`	The sowing date for each crop. The sowing date should be a real number, with `0 < nint(sowing_date) < number of days in year`. For example, for a 365 day year, sow_date = 1.0 is Jan 1st and sow_date = 365.0 is Dec 31st. If a crop requires two sowing dates per year, it should be treated as two separate crops with identical parameters apart from the sowing date. **Note:** Only required if *l_prescsow* = TRUE.
`cropttveg`	Thermal time between emergence and flowering (degree days).
`cropttrep`	Thermal time between flowering and maturity/harvest (degree days).
`croplatestharvdate`	The latest possible harvest date for each crop. croplatestharvdate is only a required variable when *l_croprotate* = TRUE and *l_prescsow* = TRUE. croplatestharvdate is not a required variable when *l_croprotate* = FALSE and *l_prescsow* = TRUE but will be used if provided in the ancillary file croplatestharvdate is not a required variable and is only used if provided as an ancillary when *l_prescsow* = TRUE.

See also:

References:

- Osborne et al, JULES-crop: a parametrisation of crops in the Joint UK Land Environment Simulator, Geosci. Model Dev., 8, 1139-1155, 2015.
- Mathison et al, 'Developing a sequential cropping capability in the JULESvn5.2 land–surface model', Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2019-85, in review, 2019

6.21.8 `JULES_IRRIG_PROPS` namelist members

This namelist specifies the options available for initialising irrigated fraction.

JULES_IRRIG_PROPS::**read_from_dump**

Type
logical

Default
F

TRUE
: Populate variables associated with this namelist from the dump file. All other namelist members are ignored.

FALSE
: Use the other namelist members to determine how to populate variables.

JULES_IRRIG_PROPS::**read_file**

Type
: logical

Default
: T

Indicates if irrigated fraction is to be read from file.

TRUE
: Irrigated fraction is read from the file specified in *irrig_frac_file*.

FALSE
: Irrigated fraction is set to the constant value specified in *const_frac_irr*.

JULES_IRRIG_PROPS::**irrig_frac_file**

Type
: character

Default
: None

The file from which irrigation fractions are read, including path.

JULES_IRRIG_PROPS::**var_name**

Type
: character

Default
: 'frac_irig'

The name of the variable containing the irrigated fraction data.

Note: This is only used for NetCDF files. For ASCII files, the irrigated fraction data is expected to be the first (ideally only) variable in the file.

In the file, the variable must have no levels or time dimensions.

JULES_IRRIG_PROPS::**const_frac_irr**

Type
: real

Default
: none

The constant irrigated fraction to be applied to all grid points.

JULES_IRRIG_PROPS::**const_irrfrac_irrtiles**

Type
: real

Default
: none

The constant irrigated fraction to be applied to specific surface tiles given in *irrigtiles*.

6.21.9 `JULES_RIVERS_PROPS` namelist members

This namelist specifies how spatially varying river routing properties should be set.

Note: `read_from_dump` is not currently implemented for this namelist, although initial condition variables can be read from a dump file if *`i_river_vn`* is `1`, `2`, or `3` (see *`JULES_INITIAL`*).

Note: The river routing code in JULES is still in development. Users should ensure that results are as expected, and provide feedback where deficiencies are identified.

Note: The grid on which the river routing will run, and on which river routing ancillaries must be provided, could potentially differ from the input/model grid specified in *model_grid.nml*.

For the duration of this document, the following nomenclature will be used:

- **Model input grid** - The full JULES input grid specified in *`JULES_INPUT_GRID`*.
- **River routing input grid** - The grid on which river routing ancillaries will be provided

Currently, information about the river routing input grid and its relationship to the model input grid is specified in *`JULES_RIVERS_PROPS`*.

While the model input can be defined on a 1D grid, the river routing input grid must be defined on a 2D grid, as defined through the x and y dimensions of the rivers ancillary file. See *`x_dim_name`* and *`y_dim_name`* for further details. If a non-regular model and river routing input grid is used, both the x and y dimensions and corresponding latitude and longitude values must be specified for each grid point.

However, internally JULES converts the river routing input grid to a 1D river routing model grid, with length `np_rivers`, which is the number of valid routing points in the river routing ancillaries. All river routing output is either defined on the 1D river routing model grid or is regridded to the model grid.

For some applications, the model input and river routing input grids may not be coincident. Note that functionality only currently exists to regrid between regular (but non-identical) model input and river routing input grids. If a non-regular model input grid is specified, it is assumed that the model input and river routing input grids will be coincident.

Members used to define the river routing input grid

`JULES_RIVERS_PROPS::`**`rivers_reglatlon`**

Type
logical

Default
T

Flag indicating if the river routing input grid is regular in latitude and longitude.

TRUE
River routing input grid is regular in latitude and longitude.

FALSE
River routing input grid is not regular in latitude and longitude (e.g. grid defined relative to a rotated pole, Ordnance Survey (British) National Grid (BNG) OSGB36, etc). Only *`i_river_vn`* `= 2` should be used in this case. Note that the model input and river routing input grids must be coincident in this case and *`rivers_regrid`* must also be set to false.

`JULES_RIVERS_PROPS::`**`coordinate_file`**

Type
character

Default
None

The file from which to read coordinate information for the river routing input grid. This is only used when `file` includes *variable-name templating*, i.e. it is only used when ancillary variables will come from multiple files, in which case this variable is used to provide clarity as to where the coordinates are read from.

`JULES_RIVERS_PROPS::`**`x_dim_name`**

Type
character

Default
None

The name of the x dimension for the river routing input grid (it may, but does not have to, coincide with *`x_dim_name`*).

Note: For ASCII files, this can be anything. For NetCDF files, it should be the name of the dimension in `file` (if that does not include *variable-name templating*) or in `coordinate_file` (if `file` includes templating).

Warning: Values for the x dimension of the river routing input grid will need to be read from the input file to define the grid, so it is assumed that the file contains a variable of the same name. If a non-regular river routing input grid is used, a 2D longitude field will also be needed to define the x-location of each grid point, read in via the longitude_2d ancillary field.

`JULES_RIVERS_PROPS::`**`y_dim_name`**

Type
character

Default
None

The name of the y dimension for the river routing input grid (it may, but does not have to, coincide with *`y_dim_name`*).

Note: For ASCII files, this can be anything. For NetCDF files, it should be the name of the dimension in `file` (if that does not include *variable-name templating*) or in `coordinate_file` (if `file` includes templating).

Warning: Values for the y dimension of the river routing input grid will need to be read from the input file to define the grid, so it is assumed that the file contains a variable of the same name. If a non-regular river routing input grid is used, a 2D latitude field will also be needed to define the y-location of each grid point, read in via the latitude_2d ancillary field.

`JULES_RIVERS_PROPS::`**`nx`**

Type
integer

Permitted
>= 1

Default
None

The size of the x dimension of the river routing input grid.

JULES_RIVERS_PROPS::**ny**

Type
integer

Permitted
>= 1

Default
None

The size of the y dimension of the river routing input grid.

Members used to define the relationship between the model input grid and the river routing input grid

JULES_RIVERS_PROPS::**rivers_regrid**

Type
logical

Default
T

Flag indicating if the model input and river routing input grids are identical, i.e. whether regridding of variables to and from the river routing input grid is required. Note this is only currently possible if *rivers_reglatlon* is TRUE.

TRUE
River routing input and model input grids differ and regridding is required.

FALSE
River routing input and model input grids are identical.

Warning: Currently, regridding between model input and river routing input grids must only be used with regular lat/lon model input and river routing input grids.

- If a 1D model input grid is specified in *JULES_INPUT_GRID*, it must be possible to define a 2D regular lat/lon grid containing all the points in the model input grid. This is done using the variables below.

An example with the GSWP2 (land points only) forcing data is given below.

Only used when JULES_INPUT_GRID::grid_is_1d = TRUE or for a parallel standalone run.

JULES_RIVERS_PROPS::**nx_grid**

Type
integer

Permitted
>= 1

Default
JULES_RIVERS_PROPS::nx

The size of the x dimension of the 2D regular lat/lon grid containing the model input grid.

JULES_RIVERS_PROPS::**ny_grid**

Type
integer

Permitted
>= 1

Default
JULES_RIVERS_PROPS::ny

The size of the y dimension of the 2D regular lat/lon grid containing the model input grid.

JULES_RIVERS_PROPS::**reg_lat1**

Type
real

Default
Latitude of lower-left corner of river routing input grid

The latitude of the lower-left corner of the 2D regular lat-lon grid containing the model input grid.

JULES_RIVERS_PROPS::**reg_lon1**

Type
real

Default
Longitude of lower-left corner of river routing input grid

The longitude of the lower-left corner of the 2D regular lat/lon grid containing the model input grid.

JULES_RIVERS_PROPS::**reg_dlat**

Type
real

Default
Latitude spacing of river routing input grid

The latitude spacing of the 2D regular lat/lon grid containing the model input grid.

JULES_RIVERS_PROPS::**reg_dlon**

Type
real

Default
Longitude spacing of river routing input grid

The longitude spacing of the 2D regular lat/lon grid containing the model input grid.

Only used when `rivers_reglatlon` = FALSE

JULES_RIVERS_PROPS::**rivers_dx**

Type
real

Permitted
> 0

Default
0

The constant size of the rivers grid (in m) if non-regular in latitude/longitude (e.g. if defined in Ordnance Survey (British) National Grid (BNG) OSGB36 coordinates).

Members used to determine how river routing variables are set

JULES_RIVERS_PROPS::**file**

Type
character

Default
None

The file to read river routing properties from.

If `use_file` is FALSE for every variable, this will not be used.

This file name can use *variable name templating*.

`JULES_RIVERS_PROPS::`**`nvars`**

Type
integer

Permitted
>= 0

Default
0

The number of river routing property variables that will be provided (see *List of rivers properties*).

- For RFM, at least direction is currently required
- For TRIP, at least direction and sequence are required

`JULES_RIVERS_PROPS::`**`var`**

Type
character(nvars)

Default
None

List of river routing variable names as recognised by JULES (see *List of rivers properties*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_RIVERS_PROPS::`**`use_file`**

Type
logical(nvars)

Default
T

For each JULES variable specified in `var`, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using `const_val` below.

`JULES_RIVERS_PROPS::`**`var_name`**

Type
character(nvars)

Default
‘’ (empty string)

For each JULES variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from `var` is used instead.

This is not used for variables where `use_file` = FALSE, but a placeholder must still be given in that case.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

`JULES_RIVERS_PROPS::tpl_name`

Type
character(nvars)

Default
None

For each JULES variable specified in *var*, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

`JULES_RIVERS_PROPS::const_val`

Type
real(nvars)

Default
None

For each JULES variable specified in *var* where *use_file* = FALSE, this is a constant value that the variable will be set to at every point.

This is not used for variables where *use_file* = TRUE, but a placeholder must still be given in that case.

Example

The following gives an example of how you would set up the namelists to use routing with the GSWP2 forcing data.

The model input grid is the GSWP2 grid, i.e. a land-points-only, 1D grid where points lie on a 1° x 1° grid. The river routing input grid is a 2D 1° x 1° grid.

Since both grids are 1° x 1°, we define the 2D regular lat-lon grid containing the model input grid to be the river routing input grid, which means we don't need any regridding of variables.

```
&JULES_INPUT_GRID
  grid_is_1d    = T,
  npoints       = 15238,
  grid_dim_name = "land"
  # ...
/

# ...

&JULES_RIVERS_PROPS
  # Define the river routing input grid to be a 2D regular lat-lon grid
  rivers_reglatlon = T,
  x_dim_name = "longitude",
  nx         = 360,
  y_dim_name = "latitude",
  ny         = 180,

  # Define the 2D regular lat-lon grid containing the model input grid to be a 2D 1\␣
↪|deg| x 1\ |deg| grid
  nx_grid  = 360,
  ny_grid  = 180,
```

(continues on next page)

(continued from previous page)

```
  reg_lat1 = -89.5,
  reg_lon1 = -179.5,
  reg_dlat = 1.0,
  reg_dlon = 1.0,

  # No regridding required since the river routing input grid is the same as the 2D␣
↪regular lat-lon grid containing the model input grid
  rivers_regrid = F
/
```

List of rivers properties

The following table summarises river routing input grid properties required to run RFM or TRIP river routing algorithms, specified from an ancillary file if *use_file* = TRUE.

Name	Description
`area`	Drainage area (in number of grid boxes) draining into a given grid box. This is used if *i_river_vn* = 2 to distinguish between river and land points using the *a_thresh* parameter. Points with drainage area > *a_thresh* are treated as rivers, all others as land. These two classes of points use different wave speeds (e.g. *cland* and *criver*). If this field is not available, all points are treated as rivers. The drainage area does not include the grid point itself, and so an extra point must be added where catchment area calculations are required.
`direction`	Flow direction for each river routing grid box, defining the next grid box into which surface or sub-surface water will be routed. This is specified as an integer according to `[1 = N, 2 = NE, 3 = E, 4 = SE, 5 = S, 6 = SW, 7 = W, 8 = NW]`. Although these are referred to via compass directions, they are used as "grid-relative" directions. e.g. "N" means "same column, one row up", "E" means "one column over, same row". Thus for a rotated grid (columns do not run S-N on the Earth), the point "same column, one row up" does does not lie immediately N. Additionally, 9: river mouth (outflow to sea) 10: inland drainage point (an endorheic catchment; no outflow from grid box) All other values (<1 or >10) are excluded from the river calculations (effectively treated as sea). Note that at present any river flow at an inland drainage point is NOT added to the soil moisture (in standalone JULES).
`sequence`	River routing network pathway number. Used by TRIP river routing only (i.e. *i_river_vn* = `1,3`). See Oki et al. (1999) for details.
`latitude_2d`	If *rivers_reglatlon* = FALSE, the unique 2D location of each river grid point must be specified.
`longitude_2d`	If *rivers_reglatlon* = FALSE, the unique 2D location of each river grid point must be specified.

See also:

References:

- Bell, V.A. et al. (2007) Development of a high resolution grid-based river flow model for use with regional climate model output. Hydrology and Earth System Sciences. 11 532-549
- Oki, T., et al (1999) Assessment of annual runoff from land surface models using Total Runoff Integrating Pathways (TRIP). Journal of the Meteorological Society of Japan. 77 235-255

6.21.10 `JULES_OVERBANK_PROPS` namelist members

This namelist specifies how the river overbank inundation properties should be set.

Note: `read_from_dump` is not currently implemented for this namelist.

Note: The grid here MUST coincide exactly with the river routing input grid specified in *JULES_RIVERS_PROPS*.

Members used to determine how overbank inundation variables are set

`JULES_OVERBANK_PROPS::`**`file`**

Type
character

Default
None

The file to read overbank inundation properties from (can be the same file as specified in *JULES_RIVERS_PROPS*).

If *`use_file`* is FALSE for every variable, this will not be used.

This file name can use *variable name templating*.

`JULES_OVERBANK_PROPS::`**`nvars`**

Type
integer

Permitted
>= 0

Default
0

The number of overbank inundation property variables that will be provided (see *List of overbank inundation properties*).

`JULES_OVERBANK_PROPS::`**`var`**

Type
character(nvars)

Default
None

List of overbank inundation variable names as recognised by JULES (see *List of overbank inundation properties*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_OVERBANK_PROPS::`**`use_file`**

Type
logical(nvars)

Default
T

For each JULES variable specified in *`var`*, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using *const_val* below.

`JULES_OVERBANK_PROPS::var_name`

Type
character(nvars)

Default
'' (empty string)

For each JULES variable specified in *var* where *use_file* = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from *var* is used instead.

This is not used for variables where *use_file* = FALSE, but a placeholder must still be given in that case.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

`JULES_OVERBANK_PROPS::tpl_name`

Type
character(nvars)

Default
None

For each JULES variable specified in *var*, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

`JULES_OVERBANK_PROPS::const_val`

Type
real(nvars)

Default
None

For each JULES variable specified in *var* where *use_file* = FALSE, this is a constant value that the variable will be set to at every point.

This is not used for variables where *use_file* = TRUE, but a placeholder must still be given in that case.

List of overbank inundation properties

The following table summarises overbank inundation grid properties, specified from an ancillary file if *use_file* = TRUE.

Name	Description
`logn_mean`	Mean of ln(elevation-elev_min) for each grid cell (in units ln(m)) This is only used if *l_riv_hypsometry* = TRUE Note that elev_min is DEM minimum, not river/lake bed level (therefore large values close to water bodies can occur in floodplain gridcells)
`logn_stdev`	Standard deviation of ln(elevation-elev_min) for each grid cell (in units ln(m)) This is only used if *l_riv_hypsometry* = TRUE

See also:

References:

- Appx B of Lewis HW, Castillo Sanchez JM, Graham J, Saulter A, Bornemann J, Arnold A, Fallmann J, Harris C, Pearson D, Ramsdale S, Martínez de la Torre A, Bricheno L, Blyth E, Bell VA, Davies H, Marthews TR, O'Neill C, Rumbold H, O'Dea E, Brereton A, Guihou K, Hines A, Butenschon M, Dadson SJ, Palmer T, Holt J, Reynard N, Best M, Edwards J & Siddorn J (2018). The UKC2 regional coupled environmental prediction system. Geoscientific Model Development 11:1-42.

6.21.11 `JULES_WATER_RESOURCES_PROPS` namelist members

This namelist specifies how the water resource ancillary properties should be set.

Members used to determine how water resource variables are set

`JULES_WATER_RESOURCES_PROPS::`**`read_from_dump`**

Type
logical

Default
F

TRUE
Populate variables associated with this namelist from the dump file. All other namelist members are ignored.

FALSE
Use the other namelist members to determine how to populate variables.

`JULES_WATER_RESOURCES_PROPS::`**`file`**

Type
character

Default
None

The file to read water resource ancillary properties from.

If *`use_file`* is FALSE for every variable, this will not be used.

This file name can use *variable name templating*.

`JULES_WATER_RESOURCES_PROPS::`**`nvars`**

Type
integer

Permitted
>= 0

Default
0

The number of water resource property variables that will be provided (see *List of water resources properties*).

`JULES_WATER_RESOURCES_PROPS::`**`var`**

Type
character(nvars)

Default
None

List of water resource variable names as recognised by JULES (see *List of water resources properties*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_WATER_RESOURCES_PROPS::use_file`

Type
logical(nvars)

Default
T

For each JULES variable specified in `var`, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using `const_val` below.

`JULES_WATER_RESOURCES_PROPS::var_name`

Type
character(nvars)

Default
'' (empty string)

For each JULES variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from `var` is used instead.

This is not used for variables where `use_file` = FALSE, but a placeholder must still be given in that case.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

`JULES_WATER_RESOURCES_PROPS::tpl_name`

Type
character(nvars)

Default
None

For each JULES variable specified in `var`, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

`JULES_WATER_RESOURCES_PROPS::const_val`

Type
real(nvars)

Default
None

For each JULES variable specified in `var` where `use_file` = FALSE, this is a constant value that the variable will be set to at every point.

This is not used for variables where `use_file` = TRUE, but a placeholder must still be given in that case.

List of water resources properties

The following table summarises ancillary fields for the water resources code, specified from an ancillary file if *use_file* = TRUE.

Name	Description
`conveyance_loss`	Fraction of water that is lost during conveyance from source to user.
`irrig_eff`	Irrigation efficiency. This is only used if *l_water_irrigation* = TRUE.
`sfc_water_frac`	Target for the fraction of demand that will be met from surface water (rather than groundwater).

6.21.12 `URBAN_PROPERTIES` namelist members

URBAN_PROPERTIES::**file**

Type
character

Default
None

The file to read urban properties from.

If *use_file* (see below) is FALSE for every variable, this will not be used.

This file name can use *variable name templating*.

URBAN_PROPERTIES::**nvars**

Type
integer

Permitted
>= 0

Default
0

The number of urban property variables that will be provided.

The required variables depend on whether MORUSES is used or not:

- If MORUSES is on, all variables must be given. However, depending on the configuration of MORUSES, not all given variables will be used. Those that will not be used could be set to constant values to avoid setting them from file.
- If MORUSES is *not* on, only `wrr` is required.

URBAN_PROPERTIES::**var**

Type
character(nvars)

Default
None

List of urban property variable names as recognised by JULES (see *List of urban properties*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

URBAN_PROPERTIES::**use_file**

Type
logical(nvars)

Default
T

For each JULES variable specified in *var*, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using *const_val* below.

URBAN_PROPERTIES::**var_name**

Type
character(nvars)

Default
'' (empty string)

For each JULES variable specified in *var* where *use_file* = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from *var* is used instead.

This is not used for variables where *use_file* = FALSE, but a placeholder must still be given.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

URBAN_PROPERTIES::**tpl_name**

Type
character(nvars)

Default
None

For each JULES variable specified in *var*, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

URBAN_PROPERTIES::**const_val**

Type
real(nvars)

Default
None

For each JULES variable specified in *var* where *use_file* = FALSE, this is a constant value that the variable will be set to at every point in every layer.

This is not used for variables where *use_file* = TRUE, but a placeholder must still be given.

List of urban properties

All of the urban property variables listed below are expected to have no levels dimensions and no time dimension.

Variable name	Description[1]	Notes
`wrr`	Repeating width ratio (or canyon fraction, W/R)	If *`l_urban_empirical`* = TRUE then this is updated with calculated values.
The following apply 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 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.

6.21.13 `JULES_CO2` namelist members

`JULES_CO2::`**`read_from_dump`**

Type
logical

Default
F

TRUE
Populate variables associated with this namelist from the dump file. All other namelist members are ignored.

FALSE
Use the other namelist members to determine how to populate variables.

`JULES_CO2::`**`co2_mmr`**

Type
real

Default
5.241e-4

Concentration of atmospheric CO2, expressed as a mass mixing ratio.

6.21.14 `JULES_FLAKE` namelist members

`JULES_FLAKE::`**`read_from_dump`**

Type
logical

Default
F

TRUE
Populate variables associated with this namelist from the dump file. All other namelist members are ignored.

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

FALSE
Use the other namelist members to determine how to populate variables.

`JULES_FLAKE::`**`file`**

Type
character

Default
None

The file to read the FLake parameters from.

`JULES_FLAKE::`**`nvars`**

Type
integer

Permitted
>= 0

Default
0

The number of FLake variables that will be provided. (At the moment lake depth is the only variable that needs to be provided).

`JULES_FLAKE::`**`var`**

Type
character(nvars)

Default
None

List of FLake parameter variable names as recognised by JULES (see *List of FLake parameters*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_FLAKE::`**`var_name`**

Type
character(nvars)

Default
'' (empty string)

For each JULES variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from `var` is used instead.

This is not used for variables where `use_file` = FALSE, but a placeholder must still be given in that case.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

`JULES_FLAKE::`**`tpl_name`**

Type
character(nvars)

Default
None

For each JULES variable specified in *var*, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

JULES_FLAKE::**use_file**

Type
logical(nvars)

Default
T

For each JULES variable specified in *var*, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using *const_val* below.

JULES_FLAKE::**const_val**

Type
real(nvars)

Default
5.0m

For each JULES variable specified in *var* where *use_file* = FALSE, this is a constant value that the variable will be set to for every point or gridbox.

This is not used for variables where *use_file* = TRUE, but a placeholder must still be given.

List of FLake parameters

Name	Description
`lake_depth`	For each gridbox, the depth of the lakes should be provided in meters. Note that for deep lakes FLake will assume an artificial lake bottom at 50m depth.

6.21.15 References for ancillaries

- Kattge, J. and Knorr, W., 2007, Temperature acclimation in a biochemical model of photosynthesis: a re-analysis of data from 36 species, Plant, Cell and Environment, 30: 1176–1190, https://doi.org/10.1111/j.1365-3040.2007.01690.x.

6.22 `cable_prognostics.nml`

This file contains a single namelist called *CABLE_PROGS* that is used to set up the initial state of prognostic variables.

6.22.1 `CABLE_PROGS` namelist members

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

`CABLE_PROGS::`**`file`**

Type
character

Default
None

The file to read initial values for CABLE prognostic variables from.

If *`use_file`* is FALSE for every variable, this will not be used.

This file name can use *variable name templating*.

`CABLE_PROGS::`**`nvars`**

Type
integer

Permitted
>= 0

Default
0

The number of prognostic variables that will be provided (see *List of CABLE prognostic variables*).

`CABLE_PROGS::`**`var`**

Type
character(nvars)

Default
None

List of CABLE prognostic variable names as recognised by CABLE (see *List of CABLE prognostic variables*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`CABLE_PROGS::`**`tpl_name`**

Type
character(nvars)

Default
None

For each CABLE variable specified in *`var`*, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

`CABLE_PROGS::`**`use_file`**

Type
logical(nvars)

Default
T

For each CABLE variable specified in *`var`*, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using `const_val` below.

`CABLE_PROGS::var_name`

Type
character(nvars)

Default
'' (empty string)

For each CABLE variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from `var` is used instead.

This is not used for variables where `use_file` = FALSE, but a placeholder must still be given in that case.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

`CABLE_PROGS::const_val`

Type
real(nvars)

Default
None

For each CABLE variable specified in `var` where `use_file` = FALSE, this is a constant value that the variable will be set to at every point in every layer.

This is not used for variables where `use_file` = TRUE, but a placeholder must still be given in that case.

List of CABLE prognostic variables

Values are set for each tile of each grid point and for each layer of soil or snow.

Name	Description
`SoilTemp_CABLE`	Temperature of each soil layer (K).
`SoilMoisture_CABLE`	Soil moisture content of each soil layer (kg m^{-2}).
`FrozenSoilFrac_CABLE`	Frozen soil moisture content of each soil layer as a fraction of saturation.
`SnowDepth_CABLE`	Depth of each snow level (m).
`SnowMass_CABLE`	Mass of each each snow level (kg).
`SnowTemp_CABLE`	Temperature for each snow layer (K).
`SnowDensity_CABLE`	Density for each snow layer (kg m^{-3}).
`SnowAge_CABLE`	Age of each snow layer
`OneLyrSnowDensity_CABLE`	Snow density when all snow treated as one layer. (kg m^{-3})
`ThreeLayerSnowFlag_CABLE`	Flag for 3 layer snow pack (0 - false, 1 - true)

6.23 `pft_params.nml`

This file sets the time and space-invariant parameters for plant functional types for the JULES land surface model. It contains one namelist called *JULES_PFTPARM*.

Note: If the crop model is on (i.e. *ncpft* > 0), the order of PFTs must be natural PFTs followed by crop PFTs.

6.23.1 `JULES_PFTPARM` namelist members

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

HCTN24 and 30 refer to Hadley Centre technical notes 24 and 30, available from the Met Office Library. For ease the direct links to these documents are:

- HCTN24 "Description of the "TRIFFID" Dynamic Global Vegetation Model"
- HCTN30 "MOSES 2.2 technical documentation"

`JULES_PFTPARM::`**`canht_ft_io`**

Type
real(npft)

Default
None

The height of each PFT (m), also known as the canopy height.

The value read here is only used if TRIFFID is not active (*l_triffid* = FALSE).

Note: If TRIFFID is active, canopy height is a prognostic variable and its initial value is read in *initial_conditions.nml*.

`JULES_PFTPARM::`**`lai_io`**

Type
real(npft)

Default
None

The leaf area index (LAI) of each PFT.

The value read here is only used if neither phenology nor TRIFFID is active (*l_phenol* = FALSE and *l_triffid* = FALSE).

Note: If phenology is active, LAI is a prognostic variable and its initial value is read in *initial_conditions.nml*. When TRIFFID is active but phenology is not active (not recommended), LAI is calculated from the canopy height (meaning that the seasonal cycle of LAI will not be correctly represented).

`JULES_PFTPARM::`**`c3_io`**

Type
integer(npft)

Default
None

Flag indicating whether PFT is C3 type.

0. Not C3 (i.e. C4).
1. C3.

`JULES_PFTPARM::orient_io`

Type
integer(npft)

Default
None

Flag indicating leaf angle distribution.

0. Spherical.
1. Horizontal.

`JULES_PFTPARM::can_struct_a_io`

Type
real(npft)

Default
None

Canopy structure factor (dimensionless). can_struct_a_io=1.0 indicates a structurally homogeneous canopy. Corresponds to the structure factor Zeta in Pinty et al 2006 except assumed not to vary with zenith angle i.e. b=0. The canopy structure factor has no effect if *can_rad_mod* = 1.

`JULES_PFTPARM::a_wl_io`

Type
real(npft)

Default
None

Allometric coefficient relating the target woody biomass to the leaf area index (kg carbon m^{-2}) (Clark et al., 2011; Table 7)

`JULES_PFTPARM::a_ws_io`

Type
real(npft)

Default
None

Woody biomass as a multiple of live stem biomass (Clark et al., 2011; Table 7).

`JULES_PFTPARM::albsnc_max_io`

Type
real(npft)

Default
None

Snow-covered albedo for large leaf area index.

Only used if *l_snow_albedo* = FALSE. See HCTN30 Eq.2.

`JULES_PFTPARM::albsnc_min_io`

Type
real(npft)

Default
None

Snow-covered albedo for zero leaf area index.

Only used if *l_snow_albedo* = FALSE. See HCTN30 Eq.2.

JULES_PFTPARM::**albsnf_max_io**

Type
real(npft)

Default
None

Snow-free albedo for large LAI.

Only used if *l_spec_albedo* = FALSE. See HCTN30 Eq.1.

JULES_PFTPARM::**albsnf_maxu_io**

Type
real(npft)

Default
None

Upper bound for the snow-free albedo for large LAI, when scaled to match input obs.

Only used if *l_spec_albedo* = FALSE and *l_albedo_obs* = TRUE.

JULES_PFTPARM::**albsnf_maxl_io**

Type
real(npft)

Default
None

Lower bound for the snow-free albedo for large LAI, when scaled to match input obs.

Only used if *l_spec_albedo* = FALSE and *l_albedo_obs* = TRUE.

JULES_PFTPARM::**alpha_io**

Type
real(npft)

Default
None

Quantum efficiency of photosynthesis (mol CO_2 per mol PAR photons).

JULES_PFTPARM::**alnir_io**

Type
real(npft)

Default
None

Leaf reflection coefficient for NIR. See HCTN30 Table 3.

Always used unless *can_rad_mod* = 1 and *l_spec_albedo* = FALSE.

JULES_PFTPARM::**alniru_io**

Type
real(npft)

Default
None

Upper limit for the leaf reflection coefficient for NIR, when *l_albedo_obs* = TRUE and when *alnir_io* is used.

JULES_PFTPARM::**alnirl_io**

Type
real(npft)

Default
None

Lower limit for the leaf reflection coefficient for NIR, when *l_albedo_obs* = TRUE and when *alnir_io* is used.

JULES_PFTPARM::**alpar_io**

Type
real(npft)

Default
None

Leaf reflection coefficient for VIS (photosyntehtically active radiation). See HCTN30 Table 3.

Always used unless *can_rad_mod* = 1 and *l_spec_albedo* = FALSE.

JULES_PFTPARM::**alparu_io**

Type
real(npft)

Default
None

Upper limit for the leaf reflection coefficient for VIS, when *l_albedo_obs* = TRUE and when *alpar_io* is used.

JULES_PFTPARM::**alparl_io**

Type
real(npft)

Default
None

Lower limit for the leaf reflection coefficient for VIS, when *l_albedo_obs* = TRUE and when *alpar_io* is used.

JULES_PFTPARM::**b_wl_io**

Type
real(npft)

Default
None

Allometric exponent relating the target woody biomass to the leaf area index. This is 5/3 in HCTN24 Eq.8. See also Clark et al. (2011, Table 7).

JULES_PFTPARM::**catch0_io**

Type
real(npft)

Default
None

Minimum canopy capacity (kg m^{-2}).

This is the minimum amount of water that can be held on the canopy. See HCTN30 p7.

JULES_PFTPARM::**dcatch_dlai_io**

Type
real(npft)

Default
None

Rate of change of canopy capacity with LAI (kg m^{-2}).

Canopy capacity is calculated as `catch0 + dcatch_dlai*lai`. See HCTN30 p7.

`JULES_PFTPARM::dgl_dm_io`

Type
real(npft)

Default
None

Rate of change of leaf turnover rate with moisture availability.

`JULES_PFTPARM::dgl_dt_io`

Type
real(npft)

Default
None

Rate of change of leaf turnover rate with temperature (K^{-1}).

This is 9 in HCTN24 Eq.10.

`JULES_PFTPARM::dqcrit_io`

Type
real(npft)

Default
None

Critical humidity deficit (kg H_2O per kg air).

Only used with the Jacobs model of stomatal conductance (*stomata_model* = 1).

`JULES_PFTPARM::dz0v_dh_io`

Type
real(npft)

Default
None

Rate of change of vegetation roughness length for momentum with height.

Roughness length is calculated as `dz0v_dh * canht_ft`. See HCTN30 p5.

Used if logical *l_spec_veg_z0* is set to .false.

`JULES_PFTPARM::z0v_io`

Type
real(npft)

Default
None

Specified values for the vegetation roughness length for momentum.

Used if logical *l_spec_veg_z0* is set to .true.

`JULES_PFTPARM::eta_sl_io`

Type
real(npft)

Default
None

Live stemwood coefficient (kg C/m/(m2 leaf)) (Clark et al., 2011; Table 7).

`JULES_PFTPARM::fd_io`

Type
real(npft)

Default
None

Scale factor for dark respiration. See HCTN 24 Eq. 56.

`JULES_PFTPARM::fsmc_of_io`

Type
real(npft)

Default
None

Moisture availability below which leaves are dropped.

`JULES_PFTPARM::f0_io`

Type
real(npft)

Default
None

`CI / CA` for `DQ = 0`. See HCTN 24 Eq. 32.

Only used with the Jacobs model of stomatal conductance (*stomata_model* = 1).

`JULES_PFTPARM::g1_stomata_io`

Type
real(npft)

Default
None

Parameter g1 for the Medlyn et al. (2011) model of stomatal conductance ($kPa^{0.5}$) - this is the sensitivity of the stomatal conductance to the assimilation rate. See Eqn.11 in Medlyn et al. (2012), https://doi.org/10.1111/j.1365-2486.2012.02790.x.

Only used with the Medlyn model of stomatal conductance (*stomata_model* = 2).

`JULES_PFTPARM::g_leaf_0_io`

Type
real(npft)

Default
None

Minimum turnover rate for leaves (/360days).

`JULES_PFTPARM::glmin_io`

Type
real(npft)

Default
None

Minimum leaf conductance for H_2O ($m\ s^{-1}$).

`JULES_PFTPARM::infil_f_io`

Type
real(npft)

Default
None

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 PFT-dependent factors, such as macro-pores related to vegetation roots.

See HCTN30 p14 for full details.

`JULES_PFTPARM::gsoil_f_io`

Type
real(npft)

Default
None

Soil conductance enhancement factor.

The soil conductance for soil under a PFT canopy may be modified for each PFT (as compared to the bare soil conductance) to account for PFT-dependent factors.

`JULES_PFTPARM::hw_sw_io`

Type
real(npft)

Default
None

Ratio of N stem to N heartwood (kgN/kgN) from the TRY database.

Only used if *`l_trait_phys`* = T.

`JULES_PFTPARM::kext_io`

Type
real(npft)

Default
None

Light extinction coefficient - used with Beer's Law for light absorption through plant canopies. See HCTN30 Eq.3.

`JULES_PFTPARM::kpar_io`

Type
real(npft)

Default
None

PAR Extinction coefficient (m^2 leaf / m^2 ground).

`JULES_PFTPARM::lai_alb_lim_io`

Type
real(npft)

Default
None

Minimum LAI permitted in calculation of the albedo in snow-free conditions.

`JULES_PFTPARM::neff_io`

Type
real(npft)

Default
None

Scale factor relating V_{cmax} with leaf nitrogen concentration. See HCTN 24 Eq. 51.

Only used if *l_trait_phys* = F.

`JULES_PFTPARM::nl0_io`

Type
real(npft)

Default
None

Top leaf nitrogen concentration (kg N/kg C).

Only used if *l_trait_phys* = F.

`JULES_PFTPARM::nr_nl_io`

Type
real(npft)

Default
None

Ratio of root nitrogen concentration to leaf nitrogen concentration.

`JULES_PFTPARM::nr_io`

Type
real(npft)

Default
None

Root nitrogen concentration (kgN/kgC). Only used if *l_trait_phys* = T.

`JULES_PFTPARM::ns_nl_io`

Type
real(npft)

Default
None

Ratio of stem nitrogen concentration to leaf nitrogen concentration.

`JULES_PFTPARM::nsw_io`

Type
real(npft)

Default
None

Stemwood nitrogen concentration (kgN/kgC). Only used if *l_trait_phys* = T.

`JULES_PFTPARM::hw_sw_io`

Type
real(npft)

Default
None

Ratio of Heartwood to Stemwood Nitrogen Concentration (typically 0.5) Only used if *l_trait_phys* = T.

`JULES_PFTPARM::omega_io`

Type
real(npft)

Default
None

Leaf scattering coefficient for PAR.

Always used unless *can_rad_mod* = 1 and *l_spec_albedo* = FALSE.

`JULES_PFTPARM::omegau_io`

Type
real(npft)

Default
None

Upper limit for the leaf scattering coefficient for PAR, when *l_albedo_obs* = TRUE and when *omega_io* is used.

`JULES_PFTPARM::omegal_io`

Type
real(npft)

Default
None

Lower limit for the leaf scattering coefficient for PAR, when *l_albedo_obs* = TRUE and when *omega_io* is used.

`JULES_PFTPARM::omnir_io`

Type
real(npft)

Default
None

Leaf scattering coefficient for NIR.

Always used unless *can_rad_mod* = 1 and *l_spec_albedo* = FALSE.

`JULES_PFTPARM::omniru_io`

Type
real(npft)

Default
None

Upper limit for the leaf scattering coefficient for NIR, when *l_albedo_obs* = TRUE and when *omnir_io* is used.

`JULES_PFTPARM::omnirl_io`

Type
real(npft)

Default
None

Lower limit for the leaf scattering coefficient for NIR, when *l_albedo_obs* = TRUE and when *omnir_io* is used.

`JULES_PFTPARM::r_grow_io`

Type
real(npft)

Default
None

Growth respiration fraction.

JULES_PFTPARM::**fsmc_mod_io**

Type
integer(npft)

Default
None

Switch for method of weighting the contribution that different soil layers make to the soil moisture availability factor fsmc.

0. (recommended) Calculate fsmc in each soil layer and take a weighted average, using the fraction of roots in each layer as weights. Root distribution e-folding depth is given by *rootd_ft_io*.
1. Calculate fsmc using average properties for the root zone. Depth of root zone is given by *rootd_ft_io*. This is not currently allowed if layered soil C (*l_layeredc* = TRUE) and the 4-pool model are selected (*soil_bgc_model* = 2) because of unplanned effects on litter inputs.

JULES_PFTPARM::**psi_open_io**

Type
real(npft)

Default
None

Soil potential above which the soil moisture stress factor on vegetation (fsmc) is one. Unit: Pa. Allowed range: must be negative. Only used if *l_use_pft_psi* = T.

JULES_PFTPARM::**psi_close_io**

Type
real(npft)

Default
None

Soil potential below which the soil moisture stress factor on vegetation (fsmc) is zero. Unit: Pa. Allowed range: must be negative. Only used if *l_use_pft_psi* = T.

JULES_PFTPARM::**rootd_ft_io**

Type
real(npft)

Default
None

Parameter determining the root depth (m).

If *fsmc_mod_io* = 0, an exponential root distribution with depth is assumed, with e-folding depth `rootd_ft` (see HCTN30 Eq.32). Note that this means that generally some of the roots exist at depths greater than `rootd_ft`. If *fsmc_mod_io* = 1, `rootd_ft` is the total depth of the root zone.

JULES_PFTPARM::**fsmc_p0_io**

Type
real(npft)

Default
None

Pft-dependent parameter governing the threshold at which the plant starts to experience water stress due to lack of water in the soil. Only used if *l_use_pft_psi* = F. The volumetric soil moisture content (m^3 water per m^3 soil) at which the plant starts to become water stressed is `sm_wilt+(sm_crit-sm_wilt)*(1-fsmc_p0)` (see *JULES_SOIL_PROPS* for a description of `sm_wilt` and `sm_crit`).

JULES_PFTPARM::**sigl_io**

Type
real(npft)

Default
None

Specific density of leaf carbon (kg C/m^2 leaf) (Clark et al., 2011; Table 7).

Only used if *l_trait_phys* = F.

JULES_PFTPARM::**tleaf_of_io**

Type
real(npft)

Default
None

Temperature below which leaves are dropped (K).

JULES_PFTPARM::**tlow_io**

Type
real(npft)

Default
None

Lower temperature parameter for photosynthesis (deg C), for the Collatz model of leaf photosynthesis.

Always used for C_4 plants. Only used for C_3 plants with the Collatz model of leaf photosynthesis (*photo_model* = 1).

JULES_PFTPARM::**tupp_io**

Type
real(npft)

Default
None

Upper temperature parameter for photosynthesis (deg C), for the Collatz model of leaf photosynthesis.

Always used for C_4 plants. Only used for C_3 plants with the Collatz model of leaf photosynthesis (*photo_model* = 1).

JULES_PFTPARM::**emis_pft_io**

Type
real(npft)

Default
None

Surface emissivity of vegetated surfaces.

JULES_PFTPARM::**z0hm_pft_io**

Type
real(npft)

Default
None

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_io* and *dz0v_dh_io*.

JULES_PFTPARM::**z0hm_classic_pft_io**

Type
real(npft)

Default
None

Ratio of the roughness length for heat to the roughness length for momentum *for the CLASSIC aerosol scheme only*.

Note: This makes no difference to the model when running standalone, and is only required to keep the standalone and UM interfaces consistent.

JULES_PFTPARM::**fl_o3_ct_io**

Type
real(npft)

Default
None

Critical flux of O3 to vegetation (nmol m^{-2} s^{-1}).

JULES_PFTPARM::**dfp_dcuo_io**

Type
real(npft)

Default
None

Plant type specific O3 sensitivity parameter ($nmol^{-1}$ m^2 s).

JULES_PFTPARM::**ief_io**

Type
real(npft)

Default
None

Isoprene Emission Factor (µg g^{-1} h^{-1}).

JULES_PFTPARM::**tef_io**

Type
real(npft)

Default
None

Monoterpene Emission Factor (µg g^{-1} h^{-1}).

JULES_PFTPARM::**mef_io**

Type
real(npft)

Default
None

Methanol Emission Factor (µg g^{-1} h^{-1}).

JULES_PFTPARM::**aef_io**

Type
real(npft)

Default
None

Acetone Emission Factor (μg g^{-1} h^{-1}).

`JULES_PFTPARM::ci_st_io`

Tybe
real(npft)

Default
None

Leaf-internal CO$_2$concentration at standard conditions (Pa),

Note: Standard conditions are: T = 303.15K, p = 1013.25 hPa, atmospheric CO$_2$ = 370 ppmv, PAR = 1000 μmol m^{-2} s^{-1}.

`JULES_PFTPARM::gpp_st_io`

Tybe
real(npft)

Default
None

Gross primary production (GPP) at standard conditions (kgC m^{-2} s^{-1}),

Note: Standard conditions are: T = 303.15K, p = 1013.25 hPa, atmospheric CO$_2$ = 370 ppmv, PAR = 1000 μmol m^{-2} s^{-1}.

`JULES_PFTPARM::nmass_io`

Type
real(npft)

Default
None

Top leaf nitrogen content per unit mass (kgN kgLeaf^{-1}).

Only used if *l_trait_phys* = T.

`JULES_PFTPARM::lma_io`

Type
real(npft)

Default
None

Leaf mass per unit area (kgLeaf m^{-2}).

Only used if *l_trait_phys* = T.

`JULES_PFTPARM::vint_io`

Type
real(npft)

Default
None

There is a linear relationship between Vcmax and Narea. Previously Vcmax was calculated as the product of nl0 and neff.

This is now replaced by a linear regression based on data reported in Kattge et al. 2009. Vint is the y-intercept, vsl is the slope.

Units: µmol CO_2 m^{-2} s^{-1}.

Only used if *l_trait_phys* = T.

JULES_PFTPARM::**vsl_io**

Type
real(npft)

Default
None

Slope in the linear regression between Vcmax and Narea.

Units: µmol CO_2 gN^{-1} s^{-1}.

Only used if *l_trait_phys* = T.

JULES_PFTPARM::**kn_io**

Type
real(npft)

Default
None.

Parameter for decay of nitrogen through the canopy, as a function of layers. Only used if *can_rad_mod* = 4 or 5.

JULES_PFTPARM::**knl_io**

Type
real(npft)

Default
None.

Parameter for decay of nitrogen through the canopy, as a function of LAI. Only used if *can_rad_mod* = 6.

JULES_PFTPARM::**q10_leaf_io**

Type
real(npft)

Default
None.

Q10 factor for plant respiration.

See Cox et al. (1999) Eq. 66.

Note: Was previously a single parameter but now can have PFT-dependent values.

JULES_PFTPARM::**fef_co2_io**

Type
real(npft)

Default
None

Fire CO_2 Emission Factor (g kg^{-1}).

JULES_PFTPARM::**fef_co_io**

Type
real(npft)

Default
None

Fire CO Emission Factor (g kg^{-1}).

JULES_PFTPARM::**fef_ch4_io**

Type
real(npft)

Default
None

Fire CH_4 Emission Factor (g kg^{-1}).

JULES_PFTPARM::**fef_nox_io**

Type
real(npft)

Default
None

Fire NOx Emission Factor (g kg^{-1}).

JULES_PFTPARM::**fef_so2_io**

Type
real(npft)

Default
None

Fire SO_2 Emission Factor (g kg^{-1}).

JULES_PFTPARM::**fef_oc_io**

Type
real(npft)

Default
None

Fire OC Emission Factor (g kg^{-1}).

JULES_PFTPARM::**fef_bc_io**

Type
real(npft)

Default
None

Fire BC Emission Factor (g kg^{-1}).

JULES_PFTPARM::**ccleaf_min_io**

Type
real(npft)

Default
None

Leaf minimum combustion completeness.

JULES_PFTPARM::**ccleaf_max_io**

Type
real(npft)

Default
None

Leaf maximum combustion completeness.

JULES_PFTPARM::**ccwood_min_io**

Type
real(npft)

Default
None

Wood minimum combustion completeness.

JULES_PFTPARM::**ccwood_max_io**

Type
real(npft)

Default
None

Wood maximum combustion completeness.

JULES_PFTPARM::**avg_ba_io**

Type
real(npft)

Default
None

Average PFT Burnt Area per fire (m^2).

JULES_PFTPARM::**fire_mort_io**

Type
real(npft)

Default
None

Scaling factor for vegetation mortality caused by fire (from INFERNO burned area). Can be varied between 0.0 (no morality) and 1.0 (100% mortality) for each PFT.

See also:

References:

- Clark et al., 2011, The Joint UK Land Environment Simulator (JULES), model description – Part 2: Carbon fluxes and vegetation dynamics, Geosci. Model Dev., 4, 701-722, https://doi.org/10.5194/gmd-4-701-2011
- Pinty, B., T. Lavergne, R. E. Dickinson, J.-L. Widlowski, N. Gobron, and M. M. Verstraete (2006), Simplifying the interaction of land surfaces with radiation for relating remote sensing products to climate models, J. Geophys. Res., 111, D02116, https://doi.org/10.1029/2005JD005952.

Only used with the Farquhar model of leaf photosynthesis (`photo_model` = 2). A value is required for each PFT, but only those for C_3 plants are used (since only C_3 plants use the Farquhar model). Below, J_{max} is the potential rate of electron transport, and V_{cmax} is the maximum rate of carboxylation of Rubisco.

JULES_PFTPARM::**act_jmax_io**

Type
real(npft)

Default
None

Activation energy for temperature response of J_{max} (J mol^{-1}).

JULES_PFTPARM::**act_vcmax_io**

Type
real(npft)

Default
None

Activation energy for temperature response of V_{cmax} (J mol^{-1}).

Note: *act_jmax_io* and *act_vcmax_io* are NOT required if thermal adaptation or acclimation of photosynthesis is selected (*photo_acclim_model* = 1, 2 or 3) together with *photo_act_model* = 2.

`JULES_PFTPARM::alpha_elec_io`

Type
real(npft)

Default
None

Quantum yield of electron transport (mol electrons [mol^{-1} PAR photons]).

`JULES_PFTPARM::deact_jmax_io`

Type
real(npft)

Default
None

Deactivation energy for temperature response of J_{max} (J mol^{-1}). This describes the rate of decrease above the optimum temperature.

`JULES_PFTPARM::deact_vcmax_io`

Type
real(npft)

Default
None

Deactivation energy for temperature response of V_{cmax} (J mol^{-1}). This describes the rate of decrease above the optimum temperature.

`JULES_PFTPARM::jv25_ratio_io`

Type
real(npft)

Default
None

Ratio of J_{max} to V_{cmax} at 25 deg C (mol electrons [mol^{-1} CO_2]).

Note: If thermal adaptation or acclimation of photosynthesis is selected (*photo_acclim_model* = 1 or 2) together with *photo_jv_model* =2 (J_{max}/V_{cmax} calculated assuming constant total nitrogen allocation)), this value is used along with parameters *n_alloc_jmax* and *n_alloc_vcmax* to calculate the final value of J_{max}/V_{cmax}.

Only used if thermal adaptation or acclimation of photosynthetic capacity is NOT modelled (`photo_acclim_model` = 0). A value is required for each PFT, but only those for C_3 plants are used (since only C_3 plants use the Farquhar model).

`JULES_PFTPARM::ds_jmax_io`

Type
real(npft)

Default
None

Entropy factor for temperature reponse of J_{max} (J mol^{-1} K^{-1}).

`JULES_PFTPARM::`**`ds_vcmax_io`**

Type
real(npft)

Default
None

Entropy factor for temperature reponse of V_{cmax} (J mol^{-1} K^{-1}).

6.24 `cable_pftparm.nml`

This file sets the time and space-invariant parameters for plant functional types for the CABLE land surface model. It contains one namelist called *`CABLE_PFTPARM`*.

6.24.1 `CABLE_PFTPARM` namelist members

This namelist reads the values of parameters for each of the plant functional types (PFTs) if the CABLE land surface model is being used. These parameters are a function of PFT only. Every member must be given a value for every run. CABLE uses the same parameters for veg and non-veg surface types, unlike JULES, and therefore its arrays are of dimension (npft + nnvg).

`CABLE_PFTPARM::`**`a1gs_io`**

Type
real(npft + nnvg)

Default
MDI

Represents the sensitivity of stomatal conductance to the assimilation rate (unitless).

`CABLE_PFTPARM::`**`alpha_io`**

Type
real(npft + nnvg)

Default
MDI

Initial slope of J-Q response curve. Units: mol (electrons) mol^{-1} (photons) (C3) mol (CO_2) mol^{-1} (photons) (C4)

`CABLE_PFTPARM::`**`canst1_io`**

Type
real(npft + nnvg)

Default
MDI

Maximum intercepted water by canopy. (mm LAI^{-1})

`CABLE_PFTPARM::`**`cfrd_io`**

Type
real(npft + nnvg)

Default
MDI

Ratio of day respiration to vcmax

`CABLE_PFTPARM::clitt_io`

Type
real(npft + nnvg)

Default
MDI

Leaf litter (alters resistance to soil evaporation) (tC ha^{-1})

`CABLE_PFTPARM::conkc0_io`

Type
real(npft + nnvg)

Default
MDI

Michaelis-menton constant for carboxylase (bar)

`CABLE_PFTPARM::conko0_io`

Type
real(npft + nnvg)

Default
MDI

Michaelis-menton constant for oxygenase (bar)

`CABLE_PFTPARM::convex_io`

Type
real(npft + nnvg)

Default
MDI

Convexity of J-Q response curve (unitless).

`CABLE_PFTPARM::cplant1_io`

Type
real(npft + nnvg)

Default
MDI

Plant carbon in 1st vegetation carbon store (g C m^{-2})

`CABLE_PFTPARM::cplant2_io`

Type
real(npft + nnvg)

Default
MDI

Plant carbon in 2nd vegetation carbon store (g C m^{-2})

`CABLE_PFTPARM::cplant3_io`

Type
real(npft + nnvg)

Default
MDI

Plant carbon in 3rd vegetation carbon store (g C m^{-2})

`CABLE_PFTPARM::csoil1_io`

Type
real(npft + nnvg)

Default
MDI

Soil carbon in 1st soil carbon store (g C m^{-2})

`CABLE_PFTPARM::csoil2_io`

Type
real(npft + nnvg)

Default
MDI

Soil carbon in 2nd soil carbon store (g C m^{-2})

`CABLE_PFTPARM::d0gs_io`

Type
real(npft + nnvg)

Default
MDI

d0 in stomatal conductance model (kPa)

`CABLE_PFTPARM::ejmax_io`

Type
real(npft + nnvg)

Default
MDI

Maximum potential electron transport rate top leaf, currently double the assigned value of vcmax. (mol m^{-2} s^{-1})

`CABLE_PFTPARM::ekc_io`

Type
real(npft + nnvg)

Default
MDI

Activation energy for carboxylase (J mol^{-1})

`CABLE_PFTPARM::eko_io`

Type
real(npft + nnvg)

Default
MDI

Activation energy for oxygenase (J mol^{-1})

`CABLE_PFTPARM::extkn_io`

Type
real(npft + nnvg)

Default
MDI

Extinction coefficient for vertical profile of N.

`CABLE_PFTPARM::frac4_io`

Type
real(npft + nnvg)

Default
MDI

Fraction of c4 plants

`CABLE_PFTPARM::froot1_io`

Type
real(npft + nnvg)

Default
MDI

Fraction of root in 1st soil layer.

`CABLE_PFTPARM::froot2_io`

Type
real(npft + nnvg)

Default
MDI

Fraction of root in 2nd soil layer.

`CABLE_PFTPARM::froot3_io`

Type
real(npft + nnvg)

Default
MDI

Fraction of root in 3rd soil layer.

`CABLE_PFTPARM::froot4_io`

Type
real(npft + nnvg)

Default
MDI

Fraction of root in 4th soil layer.

`CABLE_PFTPARM::froot5_io`

Type
real(npft + nnvg)

Default
MDI

Fraction of root in 5th soil layer.

`CABLE_PFTPARM::froot6_io`

Type
real(npft + nnvg)

Default
MDI

Fraction of root in 6th soil layer.

`CABLE_PFTPARM::g0_io`

Type
real(npft + nnvg)

Default
MDI

Residual stomatal conductance as net assimilation rate reaches zero (mol m^{-2} s^{-1})

`CABLE_PFTPARM::g1_io`

Type
real(npft + nnvg)

Default
MDI

Sensitivity of stomatal conductance to the assimilation rate (kPa).

`CABLE_PFTPARM::gswmin_io`

Type
real(npft + nnvg)

Default
MDI

Minimal stomatal conductance (mol m^{-2} s^{-1})

`CABLE_PFTPARM::hc_io`

Type
real(npft + nnvg)

Default
MDI

Height of canopy (m)

`CABLE_PFTPARM::lai_io`

Type
real(npft + nnvg)

Default
None

The leaf area index (LAI) of each PFT.

`CABLE_PFTPARM::length_io`

Type
real(npft + nnvg)

Default
MDI

Leaf length (m)

`CABLE_PFTPARM::ratecp1_io`

Type
real(npft + nnvg)

Default
MDI

Plant carbon pool rate constant in 1st vegetation carbon store (year^{-1}).

CABLE_PFTPARM::**ratecp2_io**

Type
real(npft + nnvg)

Default
MDI

Plant carbon pool rate constant in 2nd vegetation carbon store (year^{-1}).

CABLE_PFTPARM::**ratecp3_io**

Type
real(npft + nnvg)

Default
MDI

Plant carbon pool rate constant in 3rd vegetation carbon store (year^{-1}).

CABLE_PFTPARM::**ratecs1_io**

Type
real(npft + nnvg)

Default
MDI

Soil carbon pool rate constant in 1st soil carbon store (year^{-1}).

CABLE_PFTPARM::**ratecs2_io**

Type
real(npft + nnvg)

Default
MDI

Soil carbon pool rate constant in 2nd soil carbon store (year^{-1}).

CABLE_PFTPARM::**refl1_io**

Type
real(npft + nnvg)

Default
MDI

Leaf reflectance in 1st radiation band.

CABLE_PFTPARM::**refl2_io**

Type
real(npft + nnvg)

Default
MDI

Leaf reflectance in 2nd radiation band.

CABLE_PFTPARM::**refl3_io**

Type
real(npft + nnvg)

Default
MDI

Leaf reflectance in 3rd radiation band.

`CABLE_PFTPARM::rp20_io`

Type
real(npft + nnvg)

Default
MDI

Plant respiration scaler

`CABLE_PFTPARM::rpcoef_io`

Type
real(npft + nnvg)

Default
MDI

Temperature coefficient for non-leaf plant respiration (C^{-1})

`CABLE_PFTPARM::rs20_io`

Type
real(npft + nnvg)

Default
MDI

Soil respiration at 20 deg C

`CABLE_PFTPARM::shelrb_io`

Type
real(npft + nnvg)

Default
MDI

Sheltering factor

`CABLE_PFTPARM::taul1_io`

Type
real(npft + nnvg)

Default
MDI

Leaf transmittance in 1st radiation band

`CABLE_PFTPARM::taul2_io`

Type
real(npft + nnvg)

Default
MDI

Leaf transmittance in 2nd radiation band

`CABLE_PFTPARM::taul3_io`

Type
real(npft + nnvg)

Default
MDI

Leaf transmittance in 3rd radiation band

`CABLE_PFTPARM::`**`tmaxvj_io`**

Type
real(npft + nnvg)

Default
MDI

Maximum temperature of the start of photosynthesis (deg C)

`CABLE_PFTPARM::`**`tminvj_io`**

Type
real(npft + nnvg)

Default
MDI

Minimum temperature of the start of photosynthesis (deg C)

`CABLE_PFTPARM::`**`vbeta_io`**

Type
real(npft + nnvg)

Default
MDI

Stomatal sensitivity to soil water.

`CABLE_PFTPARM::`**`vcmax_io`**

Type
real(npft + nnvg)

Default
MDI

Maximum RuBP carboxylation rate top leaf. (mol m^{-2} s^{-1})

`CABLE_PFTPARM::`**`vegcf_io`**

Type
real(npft + nnvg)

Default
MDI

Scalar on soil respiration (place-holder scheme)

`CABLE_PFTPARM::`**`wai_io`**

Type
real(npft + nnvg)

Default
MDI

Wood area index (stem + branches + twigs) (not currently used in any calculations)

`CABLE_PFTPARM::`**`width_io`**

Type
real(npft + nnvg)

Default
MDI

Leaf width (m)

`CABLE_PFTPARM::`**`xalbnir_io`**

Type
real(npft + nnvg)

Default
MDI

Not currently used in any calculations.

`CABLE_PFTPARM::`**`xfang_io`**

Type
real(npft + nnvg)

Default
MDI

Leaf angle parameter

`CABLE_PFTPARM::`**`zr_io`**

Type
real(npft + nnvg)

Default
MDI

Maximum rooting depth (cm)

6.25 `nveg_params.nml`

This file contains a namelist called *`JULES_NVEGPARM`* that sets time-invariant parameters for non-vegetation surface types for the JULES land surface model.

6.25.1 `JULES_NVEGPARM` namelist members

This namelist reads the values of parameters for each of the non-vegetation surface types if the JULES land surface model is being used. These parameters are a function of surface type only. All parameters must be defined for any configuration.

HCTN30 refers to Hadley Centre technical note 30, available from the Met Office Library.

`JULES_NVEGPARM::`**`albsnc_nvg_io`**

Type
real(nnvg)

Default
None

Snow-covered albedo.

Only used if *`l_snow_albedo`* = FALSE. See HCTN30 Table 1.

`JULES_NVEGPARM::`**`albsnf_nvg_io`**

Type
real(nnvg)

Default
None

Snow-free albedo.

See HCTN30 Table 1.

A bare soil snow-free albedo of -1.0 indicates that it is supplied by an ancillary field.

`JULES_NVEGPARM::`**`albsnf_nvgu_io`**

Type
real(nnvg)

Default
None

Upper limit on snow-free albedo, when *`l_albedo_obs`* = TRUE.

`JULES_NVEGPARM::`**`albsnf_nvgl_io`**

Type
real(nnvg)

Default
None

Lower limit on snow-free albedo, when *`l_albedo_obs`* = TRUE.

`JULES_NVEGPARM::`**`catch_nvg_io`**

Type
real(nnvg)

Default
None

Capacity for water (kg m^{-2}).

See HCTN30 p7.

`JULES_NVEGPARM::`**`gs_nvg_io`**

Type
real(nnvg)

Default
None

Surface conductance (m s^{-1}).

See HCTN30 p7. Soil conductance is modified by soil moisture according to HCTN30 Eq35.

`JULES_NVEGPARM::`**`infil_nvg_io`**

Type
real(nnvg)

Default
None

Infiltration enhancement factor.

The maximum infiltration rate defined by the soil parameters for the whole gridbox may be modified for each surface tile to account for tile-dependent factors.

See HCTN30 p14 for full details.

`JULES_NVEGPARM::`**`z0_nvg_io`**

Type
real(nnvg)

Default
None

Roughness length for momentum (m).

See HCTN30 Table 4.

`JULES_NVEGPARM::`**`ch_nvg_io`**

Type
real(nnvg)

Default
None

Heat capacity of this surface type (J K^{-1} m^{-2}).

Used only if *`can_model`* is 3 or 4.

`JULES_NVEGPARM::`**`vf_nvg_io`**

Type
real(nnvg)

Default
None

Fractional coverage of non-vegetation “canopy”.

Typically set to 0.0 (conductively coupled), but value of 1.0 (radiatively coupled) used if surface tile should have a heat capacity in conjunction with *`can_model`* options 3 or 4.

Note: If *`l_moruses_storage`* = T, then for the roof coupling: 0 = **uncoupled**

`JULES_NVEGPARM::`**`emis_nvg_io`**

Type
real(nnvg)

Default
None

Surface emissivity of non-vegetated surfaces.

`JULES_NVEGPARM::`**`z0hm_nvg_io`**

Type
real(nnvg)

Default
None

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_io`*.

`JULES_NVEGPARM::`**`z0hm_classic_nvg_io`**

Type
real(nnvg)

Default
None

Ratio of the roughness length for heat to the roughness length for momentum *for the CLASSIC aerosol scheme only*.

Note: This makes no difference to the model when running standalone, and is only required to keep the standalone and UM interfaces consistent.

6.26 `cable_soilparm.nml`

This file contains a namelist called *`CABLE_SOILPARM`* that sets time-invariant parameters for different soil types for the CABLE land surface model.

6.26.1 `CABLE_SOILPARM` namelist members

This namelist reads the values of parameters for each of the soil types if the CABLE land surface model is being used. These parameters are a function of surface type only. All parameters must be defined for any configuration. The number of soil types is stored in the *n_soiltypes* parameter and for the current version of CABLE is set to *9*.

`CABLE_SOILPARM::`**`silt_io`**

Type
real(n_soiltypes)

Default
MDI

Fraction of soil which is silt.

`CABLE_SOILPARM::`**`clay_io`**

Type
real(n_soiltypes)

Default
MDI

Fraction of soil which is clay.

`CABLE_SOILPARM::`**`sand_io`**

Type
real(n_soiltypes)

Default
MDI

Fraction of soil which is sand.

`CABLE_SOILPARM::`**`swilt_io`**

Type
real(n_soiltypes)

Default
MDI

Volume of H_2O at wilting (m^3 m^{-3})

`CABLE_SOILPARM::`**`sfc_io`**

Type
real(n_soiltypes)

Default
MDI

Volume of H_2O at field capacity (m^3 m^{-3})

`CABLE_SOILPARM::`**`ssat_io`**

Type
real(n_soiltypes)

Default
MDI

Volume of H_2O at saturation (m^3 m^{-3})

`CABLE_SOILPARM::`**`bch_io`**

Type
real(n_soiltypes)

Default
MDI

Parameter b in Campbell equation.

`CABLE_SOILPARM::`**`hyds_io`**

Type
real(n_soiltypes)

Default
MDI

Hydraulic conductivity at saturation (m^{-1}).

`CABLE_SOILPARM::`**`sucs_io`**

Type
real(n_soiltypes)

Default
MDI

Suction at saturation (m).

`CABLE_SOILPARM::`**`rhosoil_io`**

Type
real(n_soiltypes)

Default
MDI

Soil bulk density (kg m^{-3})

`CABLE_SOILPARM::`**`css_io`**

Type
real(n_soiltypes)

Default
MDI

Soil specific heat capacity (J kg^{-1} K^{-1}).

6.27 `crop_params.nml`

This file contains a single namelist called *`JULES_CROPPARM`* that sets time- and space-invariant parameters for each crop type.

6.27.1 JULES_CROPPARM namelist members

This namelist reads the values of parameters for each of the crop functional types. These parameters are a function of crop pft only. These parameters are only required if *ncpft* > 0. The crop pfts should be in the same order as in *pft_params.nml*.

See also:

References:

- Osborne et al, JULES-crop: a parametrisation of crops in the Joint UK Land Environment Simulator, Geosci. Model Dev., 8, 1139-1155, 2015.

Parameters introduced after the Osborne et al 2015 paper are described in the appendix of

- Williams et al, Evaluation of JULES-crop performance against site observations of irrigated maize from Mead, Nebraska, Geosci. Model Dev., 10, 1291-1320, 2017.

JULES_CROPPARM::**t_bse_io**

Type
real(ncpft)

Default
None

Base temperature (K).

JULES_CROPPARM::**t_opt_io**

Type
real(ncpft)

Default
None

Optimum temperature (K).

JULES_CROPPARM::**tmax_io**

Type
real(ncpft)

Default
None

Maximum temperature (K).

JULES_CROPPARM::**tt_emr_io**

Type
real(ncpft)

Default
None

Thermal time between sowing and emergence (deg Cd).

JULES_CROPPARM::**crit_pp_io**

Type
real(ncpft)

Default
None

Critical photoperiod (hours).

JULES_CROPPARM::**pp_sens_io**

Type
real(ncpft)

Default
None

Sensitivity of development rate to photoperiod (hours^{-1}).

`JULES_CROPPARM::rt_dir_io`

Type
real(ncpft)

Default
None

Coefficient determining relative growth of roots vertically and horizontally.

`JULES_CROPPARM::alpha1_io`

Type
real(ncpft)

Default
None

Coefficient for determining partitioning.

`JULES_CROPPARM::alpha2_io`

Type
real(ncpft)

Default
None

Coefficient for determining partitioning.

`JULES_CROPPARM::alpha3_io`

Type
real(ncpft)

Default
None

Coefficient for determining partitioning.

`JULES_CROPPARM::beta1_io`

Type
real(ncpft)

Default
None

Coefficient for determining partitioning.

`JULES_CROPPARM::beta2_io`

Type
real(ncpft)

Default
None

Coefficient for determining partitioning.

`JULES_CROPPARM::beta3_io`

Type
real(ncpft)

Default
None

Coefficient for determining partitioning.

JULES_CROPPARM::**gamma_io**

Type
real(ncpft)

Default
None

Coefficient for determining specific leaf area (m^2 kg^{-1}).

JULES_CROPPARM::**delta_io**

Type
real(ncpft)

Default
None

Coefficient for determining specific leaf area (m^2 kg^{-1}).

JULES_CROPPARM::**remob_io**

Type
real(ncpft)

Default
None

Remobilisation factor. Fraction of stem growth partitioned to RESERVEC.

JULES_CROPPARM::**cfrac_s_io**

Type
real(npft)

Default
None

Carbon fraction of dry matter for stems.

JULES_CROPPARM::**cfrac_r_io**

Type
real(ncpft)

Default
None

Carbon fraction of dry matter for roots.

JULES_CROPPARM::**cfrac_l_io**

Type
real(ncpft)

Default
None

Carbon fraction of dry matter for leaves.

JULES_CROPPARM::**allo1_io**

Type
real(ncpft)

Default
None

Allometric coefficient relating STEMC to CANHT.

`JULES_CROPPARM::allo2_io`

Type
real(ncpft)

Default
None

Allometric coefficient relating STEMC to CANHT.

`JULES_CROPPARM::mu_io`

Type
real(ncpft)

Default
None

Allometric coefficient for calculation of senescence. MIN(mu_io * (dvi - sen_dvi_io) ** nu_io, 1.0) is the fraction of leaf carbon that is moved to the harvest pool per day once senescence has started.

`JULES_CROPPARM::nu_io`

Type
real(ncpft)

Default
None

Allometric coefficient for calculation of senescence. See description for *`mu_io`*

`JULES_CROPPARM::yield_frac_io`

Type
real(ncpft)

Default
None

Fraction of the harvest carbon pool converted to yield carbon (yield is the economically valuable component of the harvest pool e.g. kernel).

`JULES_CROPPARM::initial_carbon_io`

Type
real(ncpft)

Default
None

Carbon in crop at emergence in kgC/m2.

`JULES_CROPPARM::initial_c_dvi_io`

Type
real(ncpft)

Default
None

DVI at which the crop carbon is set to *`initial_carbon_io`*. Should be at emergence (0.0) or shortly after.

`JULES_CROPPARM::sen_dvi_io`

Type
real(ncpft)

Default
None

DVI at which leaf senescence begins.

JULES_CROPPARM::**t_mort_io**

Type
real(ncpft)

Default
None

Soil temperature (second level) at which to kill crop if DVI>1.

6.28 triffid_params.nml

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

6.28.1 JULES_TRIFFID namelist members

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

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

Note: If the crop model is on (i.e. *ncpft* > 0), only `nnpft = npft - ncpft` values will be used for each variable.

JULES_TRIFFID::**crop_io**

Type
integer(npft)

Permitted
0,1,2,3

Default
None

Flag indicating whether the PFT is natural, crop, or pasture. Only crop / pasture PFTs are allowed to grow in the agricultural area. See *l_trif_crop* for more details.

If *l_trif_crop* is FALSE permitted values of `crop_io` are 0 and 1.

0. Natural vegetation (not a crop).
1. A crop.

If *l_trif_crop* is TRUE permitted values of `crop_io` are 0, 1 and 2.

0. Natural vegetation (neither crop nor pasture).
1. Crop.
2. Pasture.

If *l_trif_biocrop* is TRUE permitted values are 0, 1, 2 and 3. Flag indicating whether the PFT is natural, crop, pasture, or bioenergy. See *l_trif_biocrop* for more details.

0. Natural vegetation.
1. Crop.
2. Pasture.
3. Bioenergy crops or trees.

`JULES_TRIFFID::g_area_io`

Type
real(npft)

Default
None

Disturbance rate (/360days).

`JULES_TRIFFID::g_grow_io`

Type
real(npft)

Default
None

Rate of leaf growth (/360days).

`JULES_TRIFFID::g_root_io`

Type
real(npft)

Default
None

Turnover rate for root biomass (/360days).

`JULES_TRIFFID::g_wood_io`

Type
real(npft)

Default
None

Turnover rate for woody biomass (/360days).

`JULES_TRIFFID::lai_max_io`

Type
real(npft)

Default
None

Maximum LAI.

`JULES_TRIFFID::lai_min_io`

Type
real(npft)

Default
None

Minimum LAI.

`JULES_TRIFFID::alloc_fast_io`

Type
real(npft)

Default
None

Fraction of the carbon flux from vegetation to wood products to add to the rapidly decaying wood products pool (wood_prod_fast).

`JULES_TRIFFID::alloc_med_io`

Type
real(npft)

Default
None

Fraction of the carbon flux from vegetation to wood products to add to the wood products pool with a moderate decay rate (wood_prod_med).

`JULES_TRIFFID::alloc_slow_io`

Type
real(npft)

Default
None

Fraction of the carbon flux from vegetation to wood products to add to the slowly decaying wood products pool (wood_prod_slow).

`JULES_TRIFFID::retran_l_io`

Type
real(npft)

Default
0.5

Fraction of retranslocated leaf N.

`JULES_TRIFFID::retran_r_io`

Type
real(npft)

Default
0.2

Fraction of retranslocated root N.

`JULES_TRIFFID::ag_expand_io`

Type
integer(npft)

Permitted
0,1

Default
0

Only used if *l_ag_expand* = TRUE.

0. No automatic expansion of PFT area when the agricultural area increases.
1. Automatically plant out new crop areas with the selected PFT.

Only used when `l_trif_biocrop` = TRUE

`JULES_TRIFFID::harvest_type_io`

Type
integer(npft)

Permitted
0,1,2

Default
0

0. No harvest (default).
1. Continuous harvest from litter, as per *l_trif_crop*.
2. Periodic harvesting.

Note: For "natural" PFTs (*crop_io* = 0), this must be set to 0. For agricultural PFTs, this can be set to 0, 1 or 2.

Only used when `harvest_type_io` = 2

A placeholder value must be used for all other PFTs.

JULES_TRIFFID::**harvest_freq_io**

Type
integer(npft)

Permitted
>= 0

Default
none

Harvest freqency in years.

JULES_TRIFFID::**harvest_ht_io**

Type
real(npft)

Permitted
> 0

Default
none

Height [m] to which the PFT is reduced at each harvest cycle.

Note: *lai_min_io* must be set such that PFT height at *lai_min_io* <= *harvest_ht_io*, otherwise JULES will not start (the required value will be shown in error output).

6.29 urban.nml

This file contains one namelist called *JULES_URBAN*.

This section predominantly sets the options available for the two-tile urban scheme MORUSES. These namelists are only read if *l_urban2t*, which requires both the *urban_canyon* and *urban_roof* surface types to be used. MORUSES provides parameters for: snow free canyon albedo (*l_moruses_albedo*), canyon emissivity (*l_moruses_emissivity*), roughness length for heat (*l_moruses_rough*), roughness length for momentum (*l_moruses_macdonald*) and thermal inertia (*l_moruses_storage*). Ancillary data, predominantly required for MORUSES, is read in via *URBAN_PROPERTIES*.

For all other parameters that MORUSES does not provide, and for any MORUSES parametrisations that are turned off, values from *nveg_params.nml* will be used instead. See the switches below for more information.

See also:

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

6.29.1 `JULES_URBAN` namelist members

`JULES_URBAN::`**`anthrop_heat_scale`**

Type
real

Default
1.0

Distribution scaling factor, which allows the anthropogenic heat flux to be spread between the `urban_canyon` and `urban_roof` surface tiles such that:

- `H_roof = anthrop_heat_scale x H_canyon`
- `H_canyon x (W/R) + H_roof x ( 1.0 - W/R ) = anthrop_heat`

Has a value between 0.0 and 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.

`JULES_URBAN::`**`l_moruses_albedo`**

Type
logical

Default
F

MORUSES switch for effective canyon albedo parameterisation (snow free).

Shortwave radiative exchange in the form of an effective canyon albedo, including shading and multiple reflections, which depends on building materials, geometry and zenith angle.

TRUE
Use MORUSES parameterisation. Requires that `l_cosz` = TRUE. Also, check whether the data are provided in UTC or local solar time. To assume local solar time set `l_local_solar_time` = TRUE.

FALSE
The snow free canyon albedo is taken from `albsnf_nvg_io`.

In all cases the snow covered albedo is `albsnc_nvg_io`. MORUSES does not parameterise the roof albedo, so this is also taken from `albsnf_nvg_io`.

`JULES_URBAN::`**`l_moruses_emissivity`**

Type
logical

Default
F

MORUSES switch for effective canyon emissivity parameterisation.

Long-wave radiative exchange in the form of an effective canyon emissivity, including multiple reflections, which depends on building materials and geometry.

TRUE
Use MORUSES parameterisation.

FALSE
The canyon emissivity is taken from *emis_nvg_io*.

In either case, the roof emissivity is taken from *emis_nvg_io*.

`JULES_URBAN::l_moruses_rough`

Type
logical

Default
F

MORUSES switch for effective roughness length for heat parameterisation.

The effective roughness length for heat has a physical basis and is not calculated as a fraction of momentum. It depends on the geometry of the canyon, which affects the recirculation of the jet within the canyon. Flow within the canyon can be broken down into two regions; the recirculation and ventilation regions, where the recirculation region forms in the wake of each building. Three different flow regimes are represented:

1. Isolated roughness - Canyon has separate recirculation and ventilation regions
2. Wake interference - Recirculation region begins to impinge on the downstream building
3. Skimming flow - Recirculation region fills the entire canyon

The effective roughness length for heat is calculated using a resistance network within these regions.

TRUE
Use MORUSES parameterisation for canyon and roof.

FALSE
Values for canyon and roof are taken from *z0_nvg_io* and *z0hm_nvg_io*.

`JULES_URBAN::l_moruses_storage`

Type
logical

Default
F

MORUSES switch for thermal inertia and coupling with the underlying soil for canyon and roof.

MORUSES consists of two surfaces; a canyon (*urban_canyon*) and a roof (*urban_roof*). This MORUSES parametrisation calculates the heat capacity of each of these surface types and also modifies how they are coupled with the underlying soil. The heat capacities of the canyon and roof are calculated using the properties of the urban fabric and the geometry of the canyon. The roof has a lower thermal inertia and can respond more rapidly to changes in forcing. The nature of the coupling (radiative, conductive or none) is controlled via *vf_nvg_io* as descibed below.

The canyon: Consists of two walls and a road where the road only is coupled to the underlying soil. The walls are uncoupled and have a zero-flux boundary condition. The coupling of the road is therefore parametrised using a canyon scaling factor. The nature of the canyon (or road surface) coupling is specified as follows:

***vf_nvg_io* (*urban_canyon*):**

0	conductively coupled
1	radiatively coupled

The roof: As the roof is not in direct contact with the soil, it physically cannot be conductively coupled. It can either be radiatively coupled or uncoupled. To allow for no coupling, MORUSES modifies the code to change the meaning of conductively coupled to **NOT** coupled. The nature of the coupling is therefore specified as follows:

vf_nvg_io **(*urban_roof*):**

0	**NOT** coupled
1	radiatively coupled

TRUE
Use MORUSES parameterisation as described above.

FALSE
Values for canyon and roof are taken from *ch_nvg_io* and *vf_nvg_io* (with no modifications to coupling).

JULES_URBAN::**l_moruses_storage_thin**

Type
logical

Default
F

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 diffusivity of roofing materials.

JULES_URBAN::**l_moruses_macdonald**

Type
logical

Default
F

MORUSES switch for using MacDonald et al. (1998) to calculate effective roughness length of urban areas and displacement height from urban geometry.

TRUE
Use MacDonald et al. (1998) formulations.

FALSE
Appropriate data needs to be supplied instead.

Note: If *l_urban_empirical* = TRUE then *l_moruses_macdonald* should also be TRUE, to keep the roughness length and displacement height consistent with the morphology.

See also:

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

JULES_URBAN::**l_urban_empirical**

Type
logical

Default
F

Switch to use empirical relationships for urban geometry, based on total urban fraction. Dimensions calculated are W/R, H/W and H.

If no MORUSES parametrisations are used, i.e. the basic URBAN-2T, then only W/R is required.

If the roof fraction is not supplied i.e. canyon fraction = total urban fraction, then W/R will be used to calculate the canyon and roof fractions. W/R is also used to distribute anthropogenic heat between the roof and the canyon if *l_anthrop_heat_src* = TRUE.

TRUE
Use empirical relationships for urban geometry.

FALSE
Appropriate data needs to be supplied instead.

Warning: These are only valid for high resolutions (~1 km).

See also:

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)

6.30 `fire.nml`

This file contains a single namelist called *FIRE_SWITCHES* that sets time-invariant parameters for performing wildfire-related calculations.

6.30.1 `FIRE_SWITCHES` namelist members

`FIRE_SWITCHES::`**`l_fire`**

Type
logical

Default
F

Switch to enable the fire module.

TRUE
The fire module will be executed according to the settings of subsequent namelist members.

FALSE
The fire module will not be executed and subsequent members of the namelist will have no effect.

`FIRE_SWITCHES::`**`mcarthur_flag`**

Type
boolean

Default
F

Switch for calculating the McArthur Forest Fire Danger Index (FFDI).

`FIRE_SWITCHES::`**`mcarthur_opt`**

Type
real

Default
MDI

Switch for choosing which method of calculating the soil moisture deficit required for the McArthur Forest Fire Danger Index (FFDI). 1 uses the model soil moisture, 2 uses a fixed value of 120 mm.

`FIRE_SWITCHES::`**`canadian_flag`**

Type
boolean

Default
F

Switch for calculating the Canadian Fire Weather Index (FWI).

`FIRE_SWITCHES::`**`canadian_hemi_opt`**

Type
boolean

Default
F

If TRUE, then the month-dependent parameters used in the calculation will be offset by 6 months for the southern hemisphere. This will cause a discontinuity in results when crossing the equator.

`FIRE_SWITCHES::`**`nesterov_flag`**

Type
boolean

Default
F

Switch for calculating the Nesterov Index.

6.31 drive.nml

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

6.31.1 JULES_DRIVE namelist members

`JULES_DRIVE::`**`t_for_snow`**

Type
real

Default
None

If total precipitation is given as a forcing variable, then *t_for_snow* is the near-surface air temperature (K) at or below which the precipitation is assumed to be snowfall. At higher temperatures, all the precipitation is assumed to be liquid. The default value used to be 274.0 K.

`JULES_DRIVE::`**`t_for_con_rain`**

Type
real

Default
None

If total preciption or total rainfall are given, then `t_for_con_rain` is the near-surface air temperature (K) at or above which rainfall is assumed to be convective in origin. At lower temperatures, all the rainfall is assumed to be large-scale in origin. In this configuration all snow is assumed to be large-scale in origin. The default value used to be 373.15 K but in general this is not recommended as it effectively means all precipitation is large-scale; a value of 293.15 K might be more appropriate.

Also see `confrac`.

`t_for_con_rain` is not used if `l_point_data` = TRUE, since then there is no convective precipitation.

`JULES_DRIVE::`**`diff_frac_const`**

Type
real

Default
None

A constant value used to calculate diffuse radiation from the total downward shortwave radiation.

Only used if diffuse radiation is not given as a forcing variable (see *List of JULES forcing variables*).

Members used to control the daily disaggregator

HCTN96 refer to Hadley Centre technical note 96, available from the Met Office Library.

`JULES_DRIVE::`**`l_daily_disagg`**

Type
logical

Default
F

Switch controlling whether the disaggregator is used to convert daily data driving data to driving data at the model timestep. See HCTN96 for a description of the disaggregation methods used.

TRUE
Disaggregator is used.

> **Warning:** The disaggregator requires:
>
> 1. Daily forcing data, i.e. `data_period` = 86400
> 2. `main_run_start`, `spinup_start` and `data_start` to be 00:00:00 for some day.

FALSE
Disaggregator is not used.

`JULES_DRIVE::`**`l_disagg_const_rh`**

Type
logical

Default
F

Switch controlling sub-daily disaggregation of humidity.

Only used if `l_daily_disagg` = TRUE.

TRUE
Relative humidity is kept constant over day.

FALSE
Specific humidity is kept constant over day (apart from when limited by specific humidity at saturation).

JULES_DRIVE::**dur_conv_rain**

Type
real

Default
None

Duration of a convective rainfall event in seconds for use in the disaggregator. See HCTN96 section 2.4. A value of 21600s (6 hours) used to be the default.

Only used if *l_daily_disagg* = TRUE.

JULES_DRIVE::**dur_ls_rain**

Type
real

Default
None

Duration of a large-scale rainfall event in seconds for use in the disaggregator. See HCTN96 section 2.4. A value of 3600s (1 hour) used to be the default.

Only used if *l_daily_disagg* = TRUE.

JULES_DRIVE::**dur_conv_snow**

Type
real

Default
None

Duration of a convective snowfall event in seconds for use in the disaggregator. See HCTN96 section 2.4. A value of 3600s (1 hour) used to be the default.

Only used if *l_daily_disagg* = TRUE.

JULES_DRIVE::**dur_ls_snow**

Type
real

Default
None

Duration of a large-scale snowfall event in seconds for use in the disaggregator. See HCTN96 section 2.4. A value of 3600s (1 hour) used to be the default.

Only used if *l_daily_disagg* = TRUE.

JULES_DRIVE::**precip_disagg_method**

Type
integer

Permitted
1, 2, 3 or 4

Default
None

Switch controlling the disaggregation method for precipitation. See HCTN96 section 2.4. The default value used to be 2.

Only used if *l_daily_disagg* = TRUE.

1. Do not disaggregate precipitation.

2. Disaggregate precipitation using the method implemented in IMOGEN, which allocates the daily precipitation each type into one event of duration `dur_conv_rain`, `dur_ls_rain`, `dur_conv_snow` and `dur_ls_snow` for convective rain, large-scale rain, convective snow and large-scale snow respectively. The start time of this event is randomly distributed from the beginning of the day to the end of the day minus the event duration. If the rate of precipitation in any timestep of any type is greater than a hard-coded maximum (currently 350 mm/day), the precipitation is redistributed by the `redis` routine in IMOGEN.
3. As for 2, except no upper limit on the precipitation in a timestep.
4. The event duration variable is used to determine the fraction of wet and dry timesteps, which are then distributed randomly throughout the day.

Members used to specify perturbations to the driving data

JULES_DRIVE::**l_perturb_driving**

Type
logical

Default
F

Apply perturbation to driving data.

JULES_DRIVE::**temperature_abs_perturbation**

Type
real

Default
None

Absolute perturbation amount to add to temperature. Can be positive or negative. Only used if `l_perturb_driving` = TRUE.

JULES_DRIVE::**precip_rel_perturbation**

Type
real

Permitted
>= 0.0

Default
None

Relative perturbation for precipitation variables (a multiplicative factor). Only used if `l_perturb_driving` = TRUE.

Members used to specify `z1_tq` and `z1_uv`

JULES_DRIVE::**z1_uv_in**

Type
real

Permitted
> 0.0

Default
None

Constant value for the height (m) at which the wind data are valid for every point. This height is relative to the zero-plane, not the ground.

JULES_DRIVE::**z1_tq_vary**

Type
logical

Default
F

Switch to indicate whether `z1_tq` (the height (m) at which the temperature and humidity data are valid) should be constant for all points or spatially varying. The height is relative to the zero-plane, not the ground.

TRUE
Spatially varying `z1_tq` will be read from the file specified in *z1_tq_file*.

FALSE
`z1_tq` will be set to a constant value, specified in *z1_tq_in*, at all points.

JULES_DRIVE::**z1_tq_in**

Type
real

Permitted
> 0.0

Default
None

Constant value for `z1_tq` to be used for every point.

Only required if *z1_tq_vary* = F.

JULES_DRIVE::**z1_tq_file**

Type
character

Default
None

File to read spatially varying `z1_tq` from.

Only required if *z1_tq_vary* = T.

JULES_DRIVE::**z1_tq_var_name**

Type
character

Default
'z1_tq_in'

The name of the variable in *z1_tq_file* containing the data for `z1_tq`.

The variable should have no levels dimensions and no time dimension.

Note: This is not used for ASCII files.

However, since ASCII files can only be used for single-point runs, it is recommended to set *z1_tq_vary* = F and use *z1_tq_in* anyway.

Members used to specify boundary layer height

JULES_DRIVE::**bl_height**

Type
real

Permitted
> 0.0

Default
1000.0

Height above ground to top of the atmospheric boundary layer (m). This value is disregarded if `bl_height` is provided as prescribed data (see *List of supported variables*).

Members used to specify the start, end and period of the data

`JULES_DRIVE::`**`data_start`**

`JULES_DRIVE::`**`data_end`**

Type
character

Default
None

The times of the start of the first timestep of data and the end of the last timestep of data.

Each run of JULES (configured in *timesteps.nml*) can use part or all of the specified data. However, there must be data for all times between run start and run end (determined by `main_run_start`, `main_run_end`, `spinup_start` and `spinup_end`).

The times must be given in the format:

```
"yyyy-mm-dd hh:mm:ss"
```

`JULES_DRIVE::`**`data_period`**

Type
integer

Permitted
-2, -1 or > 0

Default
None

The period, in seconds, of the data.

Special cases:

-1: Monthly data

-2: Yearly data

Members used to specify the files containing the data

`JULES_DRIVE::`**`read_list`**

Type
logical

Default
F

Switch controlling how data file names are determined for a given time.

TRUE
Use a list of data file names with times of first data.

FALSE
Use a single data file for all times or a template describing the names of the data files.

`JULES_DRIVE::`**`nfiles`**

Type
integer

Permitted
>= 0

Default
0

Only used if *`read_list`* = TRUE.

The number of data files to read name and time of first data for.

`JULES_DRIVE::`**`file`**

Type
character

Default
None

If *`read_list`* = TRUE, this is the file to read the list of data file names and times from. Each line should be of the form:

```
'/data/file', 'yyyy-mm-dd hh:mm:ss'
```

In this case data file names may contain variable name templating only, with the proviso that either no file names use variable name templating or all file names do. The files must appear in chronological order.

If *`read_list`* = FALSE, this is either the single data file (if no templating is used) or a template for data file names. Both *time and variable name templating* may be used.

Members used to specify the provided variables

`JULES_DRIVE::`**`nvars`**

Type
integer

Permitted
>= 0

Default
0

The number of forcing variables that will be provided.

See *List of JULES forcing variables* for the available forcing variables and their possible configurations.

`JULES_DRIVE::`**`var`**

Type
character(nvars)

Default
None

List of forcing variable names as recognised by JULES (see *List of JULES forcing variables*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

JULES_DRIVE::**var_name**

Type
character(nvars)

Default
‘’ (empty string)

For each JULES variable specified in `var`, this is the name of the variable in the file(s) containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from `var` is used instead.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

JULES_DRIVE::**tpl_name**

Type
character(nvars)

Default
None

For each JULES variable specified in `var`, this is the string to substitute into the file name(s) in place of the variable name substitution string.

If the file name(s) do not use variable name templating, this is not used.

JULES_DRIVE::**interp**

Type
character(nvars)

Default
None

For each JULES variable specified in `var`, this indicates how the variable is to be interpolated in time (see *Temporal interpolation*).

List of JULES forcing variables

All of the available forcing variables listed in the sections below, are expected to have no levels dimensions, but must have a time dimension called `time_dim_name`.

Pressure, Humidity and Temperature

Name	Description
`pstar`	Air pressure (Pa).
`q`	Specific humidity (kg kg^{-1}).
`t`	Air temperature (K).

Radiation variables

The radiation forcing variables can be given in one of four ways:

`sw_down` and `lw_down`
: Downward fluxes of short- and longwave radiation are input. *This is the preferred option.*

`rad_net` and `sw_down`
: Downward shortwave and net all wavelength (downward is positive) radiation are input. The modelled albedo and surface temperature are used to calculate the downward longwave flux.

`lw_net` and `sw_net`
: Net downward fluxes of short- and longwave radiation are input. The modelled albedo and surface temperature are used to calculate the downward fluxes of shortwave and longwave radiation.

`lw_down` and `sw_net`
: Downward flux of longwave radiation and net downward flux of shortwave radiation are input. The modelled albedo is used to calculate the downward flux of shortwave radiation.

If any of the four combinations of radiation variables listed above are provided, then these are used to drive JULES. There is no default option. JULES will give a fatal error and stop if there are too many, too few or invalid forcing variables provided in the variable list.

> **Warning:** If *`l_daily_disagg`* = TRUE, then the first method must be used.

`diff_rad` can be used with any of the four methods. If it is given, diffuse radiation is input from file. If it is not given, *`diff_frac_const`* is used instead to partition the downward shortwave radiation into diffuse and direct.

Name	Description
`rad_net`	Net (all wavelength) downward radiation (W m^{-2}).
`lw_net`	Net downward longwave radiation (W m^{-2}).
`sw_net`	Net downward shortwave radiation (W m^{-2}).
`lw_down`	Downward longwave radiation (W m^{-2}).
`sw_down`	Downward shortwave radiation (W m^{-2}).
`diff_rad`	Diffuse radiation (W m^{-2}).

Precipitation variables

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

`precip`
: A single precipitation field is input. This represents the total precipitation (rainfall and snowfall). The total is partitioned between snowfall and rainfall using *`t_for_snow`*, and rainfall is then further partitioned into large-scale and convective components using *`t_for_con_rain`*. Convective snowfall is assumed to be zero.

`tot_rain` and `tot_snow`
: Two precipitation fields are input: total rainfall and total snowfall. The rainfall is partitioned between large-scale and convective, using *`t_for_con_rain`*. Convective snowfall is assumed to be zero.

`ls_rain`, `con_rain` and `tot_snow`
: Three precipitation fields are input: large-scale rainfall, convective rainfall and total snowfall. This cannot be used with *`l_point_data`* = TRUE. Convective snowfall is assumed to be zero.

`ls_rain`, `con_rain`, `ls_snow` and `con_snow`
: Four precipitation fields are input: large-scale rainfall, convective rainfall, large-scale snowfall and convective snowfall. This cannot be used with *`l_point_data`* = TRUE. Note that this is the only option that considers convective snowfall.

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

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

Warning: If *l_daily_disagg* = TRUE, then *interp* for each precipitation variable should be `f` or `nf`.

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

Wind variables

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

`wind`
The wind speed is input.

`u` and `v`
The two components of the horizontal wind (e.g. the southerly and westerly components) are input.

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

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

Daily disaggregator variables

If *l_daily_disagg* = TRUE, then the diurnal temperature range is also required:

Name	Description
`dt_range`	Diurnal temperature range (K).

6.31.2 Examples of specifying driving data

The examples below illustrate the use of some of the key settings in the namelist; other settings are omitted for clarity.

Single point ASCII driving data for one year

```
&JULES_DRIVE

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

  file = "met_data.dat",

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

  diff_frac_const = 0.1,
  t_for_con_rain  = 293.15
/
```

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

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

`sw_down` and `lw_down` are given, so the first radiation scheme (above) is used.

`precip` is not given but `tot_rain` is, so the second precipitation scheme (above) is used. *t_for_con_rain* = 293.15K means that rainfall is treated as convective in nature for temperatures at or above that threshold (not used if *l_point_data* = TRUE).

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

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

The driving data file (`met_data.dat`) should look similar to:

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

Driving data from NetCDF files with one variable per file

```
&JULES_DRIVE

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

  read_list = T,
  nfiles    = 162,

  file = "./file_list.txt",

  nvars     = 8,
  var       = 'sw_down'  'lw_down'  'tot_rain'  'tot_snow'     't'  'wind'  'pstar'   ␣
↪ 'q',
```

(continues on next page)

(continued from previous page)

```
  var_name =  'SWdown'   'LWdown'      'Rainf'      'Snowf'  'Tair'  'Wind'  'PSurf'
↪'Qair',
  tpl_name =  'SWdown'   'LWdown'      'Rainf'      'Snowf'  'Tair'  'Wind'  'PSurf'
↪'Qair',
  interp   =      'nb'       'nb'         'nb'         'nb'     'i'     'i'      'i'    ␣
↪ 'i',

  diff_frac_const = 0.1,
  t_for_con_rain  = 293.15
/
```

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

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

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

The presence of the variable name templating string in each file name shows that we are using *variable name templating*. The dates show that we do in fact have monthly files, but we cannot use time templating for these files because the start time of 03H does not conform to the requirements.

Furthermore, files for each variable are stored in separate directories. The values from *tpl_name* will be substituted into the file name templates in place of the substitution string (`%vv`). For example, pressure is held in files with names like `met_data/PSurf_data/PSurf198207.nc`, and temperature in files like `met_data/Tair_data/Tair198207.nc`.

The driving variable setup is as the previous example.

6.32 imogen.nml

This file contains three namelists called *IMOGEN_ONOFF_SWITCH*, *IMOGEN_RUN_LIST* and *IMOGEN_ANLG_VALS_LIST*. Values from this section are only used if IMOGEN is enabled. This is done via the following switch: *l_imogen* = TRUE.

Since IMOGEN calculates the forcing for an entire year at once, an IMOGEN run must have a start time of 00:00:00 on the 1st of January for some year.

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

- *nx* = 96 and *ny* = 56.

IMOGEN also uses its own I/O, so it expects IMOGEN specific files in a different format to JULES - this may change in the future. Examples of IMOGEN input files can be found in the data provided to run the 'rose stem' test suites on supported platforms (e.g. JASMIN).

See also:

References:

- Huntingford, C. and P. M. Cox (2000), An analogue model to derive additional climate change scenarios from existing GCM simulations, Climate Dynamics 16(8): 575-586. https://doi.org/10.1007/s003820000067
- Huntingford, C., et al. (2010), IMOGEN: an intermediate complexity model to evaluate terrestrial impacts of a changing climate, Geoscientific Model Development 3(2): 679-687. https://doi.org/10.5194/gmd-3-679-2010
- Comyn-Platt, E., et al. (2018), Carbon budgets for 1.5 and 2 C targets lowered by natural wetland and permafrost feedbacks, Nature Geoscience 11(8): 568-573. https://doi.org/10.1038/s41561-018-0174-9

- Zelazowski, P., et al. (2018), Climate pattern-scaling set for an ensemble of 22 GCMs–adding uncertainty to the IMOGEN version 2.0 impact system, Geoscientific Model Development 11.2: 541-560. https://doi.org/10.5194/gmd-11-541-2018

6.32.1 `IMOGEN_ONOFF_SWITCH` namelist members

`IMOGEN_ONOFF_SWITCH::`**`l_imogen`**

Type
logical

Default
F

Switch for IMOGEN.

TRUE
IMOGEN is used to generate meteorological forcing data.

FALSE
No effect.

Note: If IMOGEN is enabled, at most only *`z1_tq_vary`*, *`z1_tq_in`*, *`z1_uv_in`*, *`z1_tq_file`* and *`z1_tq_var_name`* are used from the *`JULES_DRIVE`* namelist.

6.32.2 `IMOGEN_RUN_LIST` namelist members

`IMOGEN_RUN_LIST::`**`co2_init_ppmv`**

Type
real

Default
286.085

Initial CO2 concentration (ppmv).

`IMOGEN_RUN_LIST::`**`file_scen_emits`**

Type
character

Default
None

If used, file containing CO2 emissions.

This file is expected to be in a specific format - see the IMOGEN example.

`IMOGEN_RUN_LIST::`**`file_non_co2_radf`**

Type
character

Default
None

If used, file containing non-CO2 radiative forcing values.

This file is expected to be in a specific format - see the IMOGEN example.

`IMOGEN_RUN_LIST::`**`nyr_non_co2`**

Type
integer

Default
21

Number of years for which non-co2 forcing is prescribed.

`IMOGEN_RUN_LIST::`**`file_scen_co2_ppmv`**

Type
character

Default
None

If used, file containing CO2 concentration (ppmv).

This file is expected to be in a specific format - see the IMOGEN example.

`IMOGEN_RUN_LIST::`**`ch4_init_ppbv`**

Type
real

Default
774.1

Initial CH4 concentration (ppbv).

Only if *`land_feed_ch4`* = TRUE.

`IMOGEN_RUN_LIST::`**`yr_fch4_ref`**

Type
real

Default
2000

Year for reference wetland CH4 emissions and atmospheric CH4 decay rate, i.e. *`fch4_ref`*, *`tau_ch4_ref`* & *`ch4_ppbv_ref`*.

Only if *`land_feed_ch4`* = TRUE.

`IMOGEN_RUN_LIST::`**`ch4_ppbv_ref`**

Type
real

Default
1751.02

Reference atmosphere CH4 concentration at *`yr_fch4_ref`* (ppbv).

Only if *`land_feed_ch4`* = TRUE.

`IMOGEN_RUN_LIST::`**`tau_ch4_ref`**

Type
real

Default
8.4

Lifetime of CH4 in atmosphere at *`yr_fch4_ref`* (years). Value used in Gedney et al. (2004) S3 (Table 1) from TAR, Table 4.3 (subscript d).

Only if *`land_feed_ch4`* = TRUE.

`IMOGEN_RUN_LIST::`**`fch4_ref`**

Type
real

Default
180.0

Reference wetland CH4 emissions for reference year *yr_fch4_ref* (Tg CH4/yr).

Only if *land_feed_ch4* = TRUE.

`IMOGEN_RUN_LIST::`**`file_ch4_n2o`**

Type
character

Default
None

File containing the CH4 and N2O atmos concs. The number of years in this file is defined by *nyr_ch4_n2o*. This file is expected to be an ascii file with three columns: the first column is the year, the second column is the CH4 concentration (ppbv) and the third column is the N2O concentration (ppbv). There is one row for each year and no header.

Only if *land_feed_ch4* = TRUE.

`IMOGEN_RUN_LIST::`**`nyr_ch4_n2o`**

Type
integer

Default
241

Number of years of CH4 and N2O data in *file_ch4_n2o*.

Only if *land_feed_ch4* = TRUE.

`IMOGEN_RUN_LIST::`**`anlg`**

Type
logical

Default
T

If TRUE, then use the GCM analogue model.

`IMOGEN_RUN_LIST::`**`anom`**

Type
logical

Default
T

If TRUE, then incorporate anomalies.

`IMOGEN_RUN_LIST::`**`c_emissions`**

Type
logical

Default
T

If TRUE, CO2 concentration is calculated.

`IMOGEN_RUN_LIST::`**`include_co2`**

Type
logical

Default
T

If TRUE, include adjustments to CO2 values.

`IMOGEN_RUN_LIST::`**`include_non_co2_radf`**

Type
logical

Default
T

If TRUE, include adjustments to non-CO2 radiative forcing.

`IMOGEN_RUN_LIST::`**`l_drive_with_global_temps`**

Type
logical

Default
F

If TRUE, use imogen to provide jules forcing based on the global mean temperature change and the climate patterns.

`IMOGEN_RUN_LIST::`**`land_feed_co2`**

Type
logical

Default
F

If TRUE, include land CO2 feedbacks on atmospheric CO2.

`IMOGEN_RUN_LIST::`**`land_feed_ch4`**

Type
logical

Default
F

If TRUE, include wetland CH4 feedbacks on atmospheric CH4. Prescribed CH4 concentrations assume a non-varying natural wetland CH4 component. However, when *`land_feed_ch4`* = TRUE the constant wetland CH4 emissions are perturbed using the anomaly in modelled natural wetland CH4 emission. The methane emissions are calculated for the diagnosed wetland area when *`l_top`* = TRUE. These are accumulated and passed to IMOGEN.

To ensure consistency with the observed atmospheric CH4 growth rate the model needs to be calibrated to produce *`fch4_ref`* TgCh4 per year (default 180) for the year *`yr_fch4_ref`* (default 2000). This is done by calibrating q10_ch4 (either *`q10_ch4_cs`*, *`q10_ch4_npp`*, *`q10_ch4_resps`*, depending on whether cs, npp or resps is defined as the substrate by *`ch4_substrate`*) and const_ch4 (either *`const_ch4_cs`*, *`const_ch4_npp`*, *`const_ch4_resps`*, again depending on whether cs, npp or resps is defined as the substrate by *`ch4_substrate`*). The calibration can be carried out as discussed in Comyn-Platt et al. (2018) and needs to be checked before proceeding because the model won't necessarily produce the correct values by default.

For wetland CH4 feedbacks values for the following: *`fch4_ref`*, *`tau_ch4_ref`*, *`ch4_ppbv_ref`*, *`yr_fch4_ref`*, *`ch4_init_ppbv`*, *`file_ch4_n2o`*, and *`nyr_ch4_n2o`* are also required.

See also:

References:

- Gedney, N., Cox, P. M. & Huntingford, C. Climate feedback from wetland methane emissions. Geophys. Res. Lett. 31, L20503 (2004). https://doi.org/10.1029/2004GL020919
- Comyn-Platt, E., et al. (2018), Carbon budgets for 1.5 and 2 C targets lowered by natural wetland and permafrost feedbacks, Nature Geoscience 11(8): 568-573. https://doi.org/10.1038/s41561-018-0174-9

`IMOGEN_RUN_LIST::`**`ocean_feed`**

Type
logical

Default
F

If TRUE, include ocean feedbacks on atmospheric CO2.

`IMOGEN_RUN_LIST::`**`nyr_emiss`**

Type
integer

Default
241

Number of years of emission data in file.

`IMOGEN_RUN_LIST::`**`file_points_order`**

Type
character

Default
None

File containing the mapping of IMOGEN global grid points onto IMOGEN land points (different from the JULES land points).

`IMOGEN_RUN_LIST::`**`initialise_from_dump`**

Type
logical

Default
F

Indicates how the IMOGEN prognostic variables will be initialised.

TRUE
Use a dump file (specified in *`dump_file`* below) from a previous run with IMOGEN to initialise the IMOGEN prognostics.

FALSE
IMOGEN will handle the initialisation of its prognostics internally.

`IMOGEN_RUN_LIST::`**`dump_file`**

Type
character

Default
None

The name of the dump file to initialise from.

Only used if *`initialise_from_dump`* = TRUE.

6.32.3 `IMOGEN_ANLG_VALS_LIST` namelist members

`IMOGEN_ANLG_VALS_LIST::`**`diff_frac_const_imogen`**

Type
real

Default
0.4

IMOGEN uses this instead of *`diff_frac_const`*

`IMOGEN_ANLG_VALS_LIST::`**`q2co2`**

Type
real

Default
3.74

Radiative forcing due to doubling CO2 (W m^{-2}).

`IMOGEN_ANLG_VALS_LIST::`**`f_ocean`**

Type
real

Default
0.711

Fractional coverage of the ocean.

`IMOGEN_ANLG_VALS_LIST::`**`kappa_o`**

Type
real

Default
383.8

Ocean eddy diffusivity (W m^{-1} K^{-1}).

`IMOGEN_ANLG_VALS_LIST::`**`lambda_l`**

Type
real

Default
0.52

Inverse of climate sensitivity over land (W m^{-2} K^{-1}).

`IMOGEN_ANLG_VALS_LIST::`**`lambda_o`**

Type
real

Default
1.75

Inverse of climate sensitivity over ocean (W m^{-2} K^{-1}).

`IMOGEN_ANLG_VALS_LIST::`**`mu`**

Type
real

Default
1.87

Ratio of land to ocean temperature anomalies.

IMOGEN_ANLG_VALS_LIST::**t_ocean_init**

Type
real

Default
289.28

Initial ocean temperature (K).

IMOGEN_ANLG_VALS_LIST::**dir_patt**

Type
character

Default
None

Directory containing the patterns.

Files in this directory are expected to be in a specific format - see the IMOGEN example.

IMOGEN_ANLG_VALS_LIST::**dir_clim**

Type
character

Default
None

Directory containing initialising climatology.

Files in this directory are expected to be in a specific format - see the IMOGEN example.

IMOGEN_ANLG_VALS_LIST::**dir_anom**

Type
character

Default
None

Directory containing prescribed anomalies.

Files in this directory are expected to be in a specific format - see the IMOGEN example.

6.33 prescribed_data.nml

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

6.33.1 JULES_PRESCRIBED namelist members

JULES_PRESCRIBED::**n_datasets**

Type
integer

Permitted
>= 0

Default
0

The number of datasets that will be specified using instances of the *JULES_PRESCRIBED_DATASET* namelist.

6.33.2 `JULES_PRESCRIBED_DATASET` namelist members

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

Members used to specify the start, end and period of the data

`JULES_PRESCRIBED_DATASET::`**`data_start`**

`JULES_PRESCRIBED_DATASET::`**`data_end`**

Type
character

Default
None

The times of the start of the first timestep of data and the end of the last timestep of data.

Each run of JULES (configured in *timesteps.nml*) can use part or all of the specified data. However, there must be data for all times between run start and run end (determined by `main_run_start`, `main_run_end`, `spinup_start` and `spinup_end`).

The times must be given in the format:

```
"yyyy-mm-dd hh:mm:ss"
```

`JULES_PRESCRIBED_DATASET::`**`data_period`**

Type
integer

Permitted
-2, -1 or > 0

Default
None

The period, in seconds, of the data.

Special cases:

-1: Monthly data

-2: Yearly data

`JULES_PRESCRIBED_DATASET::`**`is_climatology`**

Type
logical

Default
F

Indicates whether the data is to be used as a climatology (use the same data for every year).

TRUE
Interpret the data as a climatology. `data_start` and `data_end` must be such that exactly one year of data is specified.

FALSE
Do not interpret the data as a climatology.

Members used to specify the files containing the data

JULES_PRESCRIBED_DATASET::**read_list**

Type
logical

Default
F

Switch controlling how data file names are determined for a given time.

TRUE
Use a list of data file names with times of first data.

FALSE
Use a single data file for all times or a template describing the names of the data files.

JULES_PRESCRIBED_DATASET::**nfiles**

Type
integer

Permitted
>= 0

Default
0

Only used if *read_list* = TRUE.

The number of data files to read name and time of first data for.

JULES_PRESCRIBED_DATASET::**file**

Type
character

Default
None

If *read_list* = TRUE, this is the file to read the list of data file names and times from. Each line should be of the form:

```
'/data/file', 'yyyy-mm-dd hh:mm:ss'
```

In this case data file names may contain variable name templating only, with the proviso that either no file names use variable name templating or all file names do. The files must appear in chronological order.

If *read_list* = FALSE, this is either the single data file (if no templating is used) or a template for data file names. Both *time and variable name templating* may be used.

Members used to specify the provided variables

JULES_PRESCRIBED_DATASET::**nvars**

Type
integer

Permitted
>= 0

Default
0

The number of variables that the dataset will provide.

See *List of supported variables* for the supported variables.

`JULES_PRESCRIBED_DATASET::var`

Type
character(nvars)

Default
None

List of variable names as recognised by JULES (see *List of supported variables*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_PRESCRIBED_DATASET::var_name`

Type
character(nvars)

Default
‘’ (empty string)

For each JULES variable specified in `var`, this is the name of the variable in the file(s) containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from `var` is used instead.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

`JULES_PRESCRIBED_DATASET::tpl_name`

Type
character(nvars)

Default
None

For each JULES variable specified in `var`, this is the string to substitute into the file name(s) in place of the variable name substitution string.

If the file name(s) do not use variable name templating, this is not used.

`JULES_PRESCRIBED_DATASET::interp`

Type
character(nvars)

Default
None

For each JULES variable specified in `var`, this indicates how the variable is to be interpolated in time (see *Temporal interpolation*).

`JULES_PRESCRIBED_DATASET::prescribed_levels`

Type
integer(n) where n ranges from 1 (one level prescribed) to `sm_levels` (all levels prescribed)

Default
1, …, `sm_levels` i.e. all levels prescribed

Indices of the subset of levels to be prescribed. Currently only implemented for `var` = `sthuf` and `nvars` = 1. The numbering of the soil level indices starts at 1 (corresponding to the layer touching the surface). Note that `sthuf` data must be provided for all soil levels, but can be set to dummy values for the levels that are not prescribed.

List of supported variables

All variables input using *prescribed_data.nml* must have a time dimension using `time_dim_name`.

In theory, any variable with an entry in the subroutine `populate_var` in `model_interface_mod` (see *I/O framework*) can be updated via this mechanism, and the use of any of these variables is not explicitly prevented. However, it is up to the user to assess whether using this mechanism to update any particular variable is appropriate or desirable.

The use of the following variables is explicitly supported:

Name	Description	Levels dimension(s) required in files
`ozone`	Surface ozone concentration (ppb). **Note:** Required if *l_o3_damage* = TRUE.	None
`canht`	PFT canopy height (m). **Note:** Not possible if *l_triffid* = TRUE	Single levels dimension of size *npft* using *pft_dim_name*.
`lai`	PFT leaf area index. **Note:** Not possible if *l_triffid* = TRUE or *l_phenol* = TRUE	Single levels dimension of size *npft* using *pft_dim_name*.
`albobs_sw`	Observed SW diffuse albedo. **Note:** Required if *l_albedo_obs* = TRUE and *l_spec_albedo* = FALSE.	None
`albobs_vis`	Observed VIS diffuse albedo. **Note:** Required if *l_albedo_obs* = TRUE and *l_spec_albedo* = TRUE.	None
`albobs_nir`	Observed NIR diffuse albedo. **Note:** Required if *l_albedo_obs* = TRUE and *l_spec_albedo* = TRUE.	None
`co2_mmr`	Concentration of atmospheric CO2, expressed as a mass mixing ratio. **Note:** A single value of co2_mmr is applied globally. Data must be supplied for each gridpoint, but only the value of the first grid-point is used.	None
`sthuf`	Soil wetness for each soil layer. This is the mass of soil water (liquid and frozen), expressed as a fraction of the water content at saturation. **Note:** Soil wetness will be set to the prescribed value at the beginning of each timestep but will be incremented during that timestep. Also, it is recommended that the prescribed `sthuf` does not exceed one.	Single levels dimension of size *sm_levels* using *soil_dim_name*.
`frac_agr`	Fractional area of agricultural land in each gridbox.	None
`frac_past`	Fractional area of pasture land in each gridbox.	None
`frac_biocrop`	Fractional area of bioenergy cropland in each gridbox.	None
`tracer_field`	Surface concentration of atmospheric chemical tracers in the atmosphere, for calculation of deposition, as mass mixing ratio (kg kg $^{-1}$).	Single levels dimension of size *ndry_dep_species* using *tracer_dim_name*.
`bl_height`	Height above surface of top of atmospheric boundary layer (m).	None
`level_separation`	Separation of boundary layer levels (m). The levels are	Single levels dimension

6.34 `initial_conditions.nml`

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

6.34.1 `JULES_INITIAL` namelist members

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

`JULES_INITIAL::`**`dump_file`**

Type
logical

Default
F

Indicates whether the given *`file`* is a dump from a previous run of JULES.

TRUE
The file is a JULES dump file.

FALSE
The file is not a JULES dump file.

`JULES_INITIAL::`**`total_snow`**

Type
logical

Default
F

Switch controlling simplified initialisation of snow variables.

TRUE
Only the total mass of snow on each surface tile (see `snow_tile` in *List of initial condition variables*) 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).

FALSE
All snow variables required for the current configuration must be input separately (see *List of initial condition variables*).

Members used to set up spatially varying properties

`JULES_INITIAL::`**`file`**

Type
character

Default
None

The file to read initial conditions from.

If *`use_file`* (see below) is FALSE for every variable, this will not be used.

If *`dump_file`* = TRUE, this should be a JULES dump file.

If *`dump_file`* = FALSE, this should be a file conforming to the *JULES input requirements*. This file name may use *variable name templating*.

`JULES_INITIAL::`**`nvars`**

Type
integer

Permitted
>= 0

Default
0

The number of initial condition variables that will be provided.

See *List of initial condition variables* for those required for a particular configuration.

Note: If `dump_file` = TRUE and `nvars` = 0, then the model will attempt to initialise all required variables from the given dump file.

`JULES_INITIAL::`**`var`**

Type
character(nvars)

Default
None

List of initial condition variable names as recognised by JULES (see *List of initial condition variables*). Names are case sensitive.

Note: For ASCII files, variable names must be in the order they appear in the file.

`JULES_INITIAL::`**`use_file`**

Type
logical(nvars)

Default
T

For each JULES variable specified in `var`, this indicates if it should be read from the specified file or whether a constant value is to be used.

TRUE
The variable will be read from the file.

FALSE
The variable will be set to a constant value everywhere using `const_val` below.

`JULES_INITIAL::`**`var_name`**

Type
character(nvars)

Default
‘’ (empty string)

For each JULES variable specified in `var` where `use_file` = TRUE, this is the name of the variable in the file containing the data.

If the empty string (the default) is given for any variable, then the corresponding value from `var` is used instead.

This is not used for variables where `use_file` = FALSE, but a placeholder must still be given in that case.

Note: For ASCII files, this is not used - only the order in the file matters, as described above.

JULES_INITIAL::**tpl_name**

Type
character(nvars)

Default
None

For each JULES variable specified in *var*, this is the string to substitute into the file name in place of the variable name substitution string.

If the file name does not use variable name templating, this is not used.

JULES_INITIAL::**const_val**

Type
real(nvars)

Default
None

For each JULES variable specified in *var* where *use_file* = FALSE, this is a constant value that the variable will be set to at every point in every layer.

This is not used for variables where *use_file* = TRUE, but a placeholder must still be given.

JULES_INITIAL::**l_broadcast_soilt**

Type
logical

Default
False

Switch to allow non-soil tiled initial condition data to be broadcast to all soil tiles. This is only used when *l_tile_soil* is enabled. This helps distribute the model state, for example from a non-soil tiled run into a new run with soil tiling. Spin up of the model state should be considered when using this option.

Note that if *l_tile_soil* = TRUE and values on soil tiles are available to define the initial state (e.g. from a previous run with soil tiling), *l_broadcast_soilt* should be set to FALSE. Setting it to TRUE will result in the run failing because it will attempt to read a non-tiled variable.

List of initial condition variables

All input to the model must be on the same grid (see *Input files for JULES*), and initial conditions are no different. Even when the variable is only required for land points, values must be provided for the full input grid. Variables read as initial conditions must have no time dimension.

The variables it is possible to specify as initial conditions can be grouped into 'types' depending on the number and size of the levels dimensions they are required to have. For NetCDF files, the dimension names are those specified in the *JULES_INPUT_GRID* namelist. For variables with no type specified, no levels dimensions should be used.

The required levels dimensions for each initial condition 'type' are given in the following table:

Type	Number of levels dimensions	Levels dimension name(s)	Levels dimension size(s)
soil	1	*soil_dim_name*	*sm_levels*
pft	1	*pft_dim_name*	*npft*
cpft	1	*cpft_dim_name*	*ncpft*
type	1	*type_dim_name*	ntype (*npft* + *nnvg*)
surft	1	*tile_dim_name*	nsurft (1 if *l_aggregate* = TRUE, ntype otherwise)
sclayer	1	*sclayer_dim_name*	Number of soil biogeochemistry layers. If using the single-pool moodel (*soil_bgc_model* = 1) this is 1. If using the 4-pool model (*soil_bgc_model* = 2) with *l_layeredc* = FALSE this is 1, else with *l_layeredc* = TRUE this is equal to *sm_levels*. If using the ECOSSE model (*soil_bgc_model* = 3) this is equal to *dim_cslayer*.
scpool	2	*scpool_dim_name*, *sclayer_dim_name*	number of soil carbon pools (1 if *soil_bgc_model* = 1, 4 otherwise) and number of soil biogeochemistry layers (see sclayer above)
bedrock	1	*bedrock_dim_name*	*ns_deep* Only applicable if *l_bedrock* = TRUE
snow	2	*tile_dim_name*, *snow_dim_name*	nsurft (see above), *nsmax* Only applicable if nsmax > 0

The required variables for a particular configuration, along with their 'type' as specified above, are given in the following table.

Name	Description	Type
Always required		
canopy	Amount of intercepted water that is held on each surface tile (kg m^{-2}).	surft
cs	Soil carbon (kg m^{-2}). If using the single-pool model (*soil_bgc_model* = 1), this is the total soil carbon. Otherwise, this is the carbon in each of the 4 pools of the 4-pool or ECOSSE models.	scpool
snow_tile	If *can_model* /= 4, this is the total snow on the surface 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 surface tiles where *cansnowpft* = TRUE), snow_tile is interpreted as the snow on the canopy, except when overridden by *total_snow* = TRUE. If *total_snow* = TRUE, snow_tile is used to hold the total snow on the surface tile (and is subsequently put onto the ground at tiles that distinguish between ground and canopy stores). Further details of snow initialisation are given below.	surft
t_soil	Temperature of each soil layer (K).	soil
tstar_tile	Temperature of each surface tile (K). This is the surface or skin temperature.	surft
Required if *can_rad_mod* = 1		

continues on next page

Table 1 – continued from previous page

Name	Description	Type
gs	Surface conductance for water vapour (m s^{-1}). This is used to start the iterative calculation of gs for the first timestep only.	None
Required if sthuf is not prescribed for all levels in *JULES_PRESCRIBED*		
sthuf	Soil wetness for each soil layer. This is the mass of soil water (liquid and frozen), expressed as a fraction of the water content at saturation.	soil
Required if *l_phenol* = TRUE		
lai	Leaf area index of each PFT.	pft
Required if *l_triffid* = TRUE		
canht	Height (m) of each PFT.	pft
Required if *l_trif_biocrop* = TRUE		
years_since_harvest	Number of years since the previous harvest.	pft
Required if *l_veg_compete* = TRUE		
frac	The fraction of land area of each gridbox that is covered by each surface type. N.B. values specified here will override those at *JULES_FRAC*	type
Required if *l_irrig_dmd* = TRUE		
sthu_irr	Unfrozen soil wetness of each layer as a fraction of saturation in irrigated fraction.	soil
Required if *ncpft* > 0		
cropdvi	Development index for each crop pft.	cpft
croprootc	Root carbon pool for each crop pft (kg m^{-2}).	cpft
cropharvc	Carbon in 'harvest parts' pool for each crop pft (kg m^{-2}) .	cpft
cropreservec	Carbon in stem reserves pool for each crop pft (kg m^{-2}).	cpft
croplai	Leaf area index of each crop pft.	cpft
cropcanht	Height (m) of each crop pft.	cpft
Required if *l_top* = TRUE		
sthzw	Soil wetness in the deep LSH/TOPMODEL 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.	None
zw	Depth from the surface to the water table (m).	None
Required if *l_bedrock* = TRUE		
tsoil_deep	Temperature of each bedrock layer (K)	bedrock
Required if *l_snow_albedo* = TRUE		
rgrain	Snow surface grain size (µm) on each surface tile.	None
Required if *total_snow* = FALSE		
rho_snow	Bulk density of lying snow (kg m^{-3}). If *total_snow* = TRUE then this is set as follows: • If *nsmax* = 0, it is set to *rho_snow_const*. • If *nsmax* > 0 and there is an existing snow pack, it is set to *rho_snow_const*. • If *nsmax* > 0 and there is no snow pack, it is set to *rho_snow_fresh*.	surft
snow_depth	Depth of snow (kg m). If *total_snow* = TRUE, this is calculated from mass and density of snow.	surft
Required if *total_snow* = FALSE and *can_model* = 4		
snow_grnd	Amount of snow on the ground, beneath the canopy (kg m^{-2}), on each surface tile. If *total_snow* = TRUE this is set to snow_tile at tiles where *can_model* = 4 is active, and to zero at all other tiles.	surft

continues on next page

Table 1 – continued from previous page

Name	Description	Type
Required if *total_snow* = FALSE and *nsmax* > 0		
`nsnow`	The number of snow layers on each surface tile. If *total_snow* = TRUE this is calculated from the snow depth.	surft
`snow_ds`	Depth of snow in each layer (kg m). If *total_snow* = TRUE this is calculated from the snow depth and the number of snow layers.	snow
`snow_ice`	Mass of frozen water in each snow layer (kg m^{-2}). If *total_snow* = TRUE all snow is assumed to be ice.	snow
`snow_liq`	Mass of liquid water in each snow layer (kg m^{-2}). If *total_snow* = TRUE this is set to zero.	snow
`tsnow`	Temperature of each snow layer (K). If *total_snow* = TRUE this is set to the temperature of the top soil layer.	snow
Required if *total_snow* = FALSE, *nsmax* > 0 and *l_snow_albedo* = TRUE		
`rgrainl`	Snow grain size (μm) on each surface tile in each snow layer. If *total_snow* = TRUE this is set to `rgrain`.	snow
Required if *l_triffid* = TRUE and *l_landuse* = TRUE		
`frac_agr_prev`	Gridbox agricultural/crop fraction from previous TRIFFID timestep.	none
`wood_prod_fast`	Carbon content of the wood products pool with a fast decay rate.	none
`wood_prod_med`	Carbon content of the wood products pool with a medium decay rate.	none
`wood_prod_slow`	Carbon content of the wood products pool with a slow decay rate.	none
Required if *l_triffid* = TRUE and *l_landuse* = TRUE and *l_trif_crop* = TRUE		
`frac_past_prev`	Gridbox pasture fraction from previous TRIFFID timestep.	none
Required if *l_triffid* = TRUE and *l_landuse* = TRUE and *l_trif_biocrop* = TRUE		
`frac_biocrop_prev`	Gridbox bioenergy fraction from previous TRIFFID timestep.	none
Required if using 4-pool model (*soil_bgc_model* = 2) with *l_nitrogen* = TRUE., or if using ECOSSE (*soil_bgc_model* = 3) with *l_soil_n* = TRUE.		
`ns`	Soil nitrogen (kg m^{-2}).	scpool
Required if using 4-pool model (*soil_bgc_model* = 2) and *l_nitrogen* = TRUE.		
`n_inorg`	Soil inorganic nitrogen (kg m^{-2}).	sclayer
Required if using ECOSSE (*soil_bgc_model* = 3) and *l_soil_n* = TRUE.		
`n_amm`	Soil ammonium (kg m^{-2}).	sclayer
`n_nit`	Soil nitrate (kg m^{-2}).	sclayer
Required if *l_rivers* = TRUE, *i_river_vn* = ‘2’ and *dump_file* = TRUE		
`rfm_surfstore_rp`	Surface water storage on river routing points (m3)	none
`rfm_substore_rp`	Sub-surface water storage on river routing points (m3)	none
`rfm_flowin_rp`	Surface flow into a grid box on river routing points (m3)	none
`rfm_bflowin_rp`	Sub-surface flow into a grid box on river routing points (m3)	none
Required if *l_rivers* = TRUE, *i_river_vn* = ‘1,3’ and *dump_file* = TRUE		
`rivers_sto_rp`	Water storage (kg)	none
Required if *photo_acclim_model* = 2 or 3		
`t_growth_gb`	Running mean air temperature (K)	none

Warning:

if *l_rivers* = TRUE, *i_river_vn* = ‘2’ and *dump_file* = FALSE,
rfm_surfstore_rp, rfm_substore_rp, rfm_flowin_rp and rfm_bflowin are initialised to zero.

Warning: if *l_rivers* = TRUE, *i_river_vn* = ‘1,3’ and *dump_file* = FALSE, rivers_sto_rp is initialised to zero.

6.34.2 Examples of specification of initial state

Specification of initial state at a single point

This assumes that *l_phenol* = FALSE, *l_triffid* = FALSE, *soil_bgc_model* = 1 and *nsmax* = 0.

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

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

Or using the alternative list syntax (see *Introduction to Fortran namelists*):

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

  nvars = 8,
  var(1) = 'canopy',      use_file(1) = F,  const_val(1) = 0.0 ,
  var(2) = 'tstar_tile',  use_file(2) = F,  const_val(2) = 276.78,
  var(3) = 'cs',          use_file(3) = F,  const_val(3) = 12.1,
  var(4) = 'gs',          use_file(4) = F,  const_val(4) = 0.0,
  var(5) = 'rgrain',      use_file(5) = F,  const_val(5) = 50.0,
  var(6) = 'snow_tile',   use_file(6) = F,  const_val(6) = 0.0,
  var(7) = 'sthuf',       use_file(7) = T,
  var(8) = 't_soil',      use_file(8) = T,
/
```

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

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

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

The contents of `initial_conditions.dat` should look similar to:

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

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

Specifying initial state for gridded data using NetCDF files is similar, except that:

- *var_name* is required for each variable read from file.
- If variable name templating is used, *tpl_name* is required for each variable read from file.

Specification of initial state from an existing dump file

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

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

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

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

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

6.35 output.nml

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

6.35.1 JULES_OUTPUT namelist members

JULES_OUTPUT::**output_dir**

Type
character

Default
None

The directory used for output files. This can be an absolute or relative path.

JULES_OUTPUT::**run_id**

Type
character

Default
None

A name or identifier for the run. This is used to name output files and model dumps.

JULES_OUTPUT::**nprofiles**

Type
integer

Permitted
>= 0

Default
0

The number of output profiles that will be specified using instances of the *JULES_OUTPUT_PROFILE* namelist.

JULES_OUTPUT::**dump_period**

Type
integer

Permitted
>= 1

Default
1

The period between model dumps, unit depends on *dump_period_unit*.

In calendar year mode, the number of years between model dumps. Note that the calendar year (date) is used to determine whether a dump is written, not the number of years simulated so far. For example, a run that starts in the year 2012 and with *dump_period* = 5 will write dumps at the start of years 2015, 2020, 2025,. . .

In second of calendar day mode, the number of seconds between model dumps. Note that this is calculated using the number of seconds into the day, not the number seconds simulated so far. For example, with *dump_period_unit* = 'T' and *dump_period* = 10800, a run will write dumps at T00:00, T03:00, T06:00, T09:00, T12:00, T15:00, T18:00 and T21:00 of a calendar day.

Dumps are also written at the start and end of the main run, and at the start of each cycle of any spin up.

`JULES_OUTPUT::`**`dump_period_unit`**

Type
character(1)

Permitted
'Y', 'T'

Default
'Y'

The unit/mode for the model dump period setting *dump_period*. If *dump_period_unit* = 'Y', use calendar year mode (default) and *dump_period* is in number of years. If *dump_period_unit* = 'T', use second of calendar day mode and *dump_period* is in number of seconds.

6.35.2 `JULES_OUTPUT_PROFILE` namelist members

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

`JULES_OUTPUT_PROFILE::`**`profile_name`**

Type
character

Default
None

The name of the output profile.

This is used in file names and should be specified even if there is only one profile. The name for each profile should be unique to avoid overwriting data unintentionally.

Although any name can be used for a profile, the user may wish to choose a name that reflects the variables in the file (e.g. 'carbon', 'water') or the data frequency (e.g. 'daily', 'monthly').

`JULES_OUTPUT_PROFILE::`**`l_land_frac`**

Type
logical

Default
F

Output gridbox land fraction to output profile.

Required to add gribox land fraction to profile for example to allow JULES output to drive another JULES model.

`JULES_OUTPUT_PROFILE::`**`file_period`**

Type
integer

Permitted
-3, -2, -1 or 0

Default
0

The period for output files, i.e. the time interval during which all output goes to the same file.

Note: In all cases, output during spin-up goes into a separate file for each spin-up cycle and output during the main run goes into its own file(s).

This can be one of three values:

0
All output goes into the same file.

-1
Monthly files are produced (i.e. all output for a month goes into the same file).

-2
Annual files (calendar years) are produced.

-3
Daily files are produced.

Members used to specify the times that the profile will generate output

`JULES_OUTPUT_PROFILE::`**`output_spinup`**

Type
logical

Default
F

Determines whether the profile will provide output during model spin-up.

TRUE
Provide output during spin-up. Output is provided for the whole of the model spin-up. Output from each spin-up cycle goes into separate files.

FALSE
Do not provide any output during spin-up.

`JULES_OUTPUT_PROFILE::`**`output_main_run`**

Type
logical

Default
F

Determines whether the profile will provide output during the main model run (i.e. any part of the run after spin-up).

TRUE
Provide output during the main model run. Output will be provided for all times between *`output_start`* and *`output_end`* below.

FALSE
Do not provide any output during the main model run.

`JULES_OUTPUT_PROFILE::`**`output_initial`**

Type
logical

Default
F

Determines whether the profile will output initial data for the sections for which it is outputting.

See *Initial data* for caveats on the initial data file(s) produced.

TRUE
Output initial data for the profile.

If `output_spinup` = T, an initial data file will be output at the start of each spinup cycle.

If `output_main_run` = T and `output_start` = `main_run_start`, an initial data file will be output at the start of the main run.

FALSE
Do not output any initial data.

JULES_OUTPUT_PROFILE::**output_start**

JULES_OUTPUT_PROFILE::**output_end**

Type
character

Default
`main_run_start`, `main_run_end`

The time to start and stop collecting data for output. The times that output is actually produced are determined by `output_period` below.

If `output_period` is monthly, then `output_start` must be 00:00:00 on the 1st of some month.

If `output_period` is yearly, then `output_start` must be 00:00:00 on the 1st of January for some year.

If `output_end` is given such that an output period is not complete when the run reaches `output_end`, output will not be generated for that final period (e.g. if values are being output monthly but `output_end` is midday on the 31st December, then output will not be generated for December, even though most of December has been run).

The times must be given in the format:

```
"yyyy-mm-dd hh:mm:ss"
```

JULES_OUTPUT_PROFILE::**output_period**

Type
integer

Permitted
-2, -1 or > 1

Default
`timestep_len`

The period for output, in seconds. This controls the frequency with which output values are calculated; for time averages this is the length of time over which the average is calculated.

This must be a multiple of the timestep length, except for the special cases:

-1: Monthly period

-2: Annual period

JULES_OUTPUT_PROFILE::**sample_period**

Type
integer

Permitted
> 1

Default
timestep_len

The sampling period, in seconds, for time-averages, minima, maxima and accumulations.

This must be a factor of *output_period* and a multiple of *timestep_len*. For the 'special' cases of *output_period* (e.g. monthly and annual outputs), one day must be a mutiple of *sample_period*.

output_period controls the length of time over which any statistic (e.g. an average) is calculated; *sample_period* controls how often values are sampled within that time to construct the statistic.

Note: It is strongly recommended that *sample_period* be left at the default value for most applications.

An example of when intermittent sampling can be useful is to construct a time average of values from particular timesteps, e.g. those on which a particular sub-model is called, for a variable that is reset to zero on other timesteps. Sampling every timestep will average over both the 'good' values and the zeros, whereas intermittent sampling can be used to pick up just the 'good' values. However, this is very rarely required with JULES.

Another use for intermittent sampling is to save computational cost (at the expense of losing accuracy), though in practice this is rarely required. In exceptional cases, sampling every timestep can be relatively expensive 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.

If fields are not sampled every timestep, the averages/minima/maxima/accumulations will only be approximations.

Members used to specify the variables that the profile will output

JULES_OUTPUT_PROFILE::**nvars**

Type
integer

Permitted
>= 0

Default
0

The number of variables that the profile will provide output for.

The variables available for output are given in *JULES Output variables*.

JULES_OUTPUT_PROFILE::**var**

Type
character(nvars)

Default
None

List of variable names to output, as recognised by JULES (see *JULES Output variables*). Names are case sensitive.

JULES_OUTPUT_PROFILE::**var_name**

Type
character(nvars)

Default
‘’ (empty string)

For each variable specified in *var*, this is the name to give the variable in output files.

If the empty string (the default) is given for any variable, then the corresponding value from *var* is used instead.

JULES_OUTPUT_PROFILE::**output_type**

Type
character(nvars)

Default
‘S’

For each variable specified in *var*, this indicates the type of processing required.

Recognised values are:

S
Instantaneous or snapshot value.

M
Time mean value.

N
Time minimum value.

X
Time maximum value.

A
Accumulation over time.

For time average/minimum/maximum variables, the period over which each output value is calculated is given by *output_period*. For time-accumulation variables, *output_period* gives the period for output of an updated accumulation (i.e. how often the value is reported). For time averages, minima, maxima and accumulations, the sampling frequency is given by *sample_period*.

Note: A time-accumulation is initialised at the start of a run and thereafter accumulates until the end of the run. This may mean that accuracy is lost, particularly towards the end of long runs, if small increments are added to an already large sum. Furthermore, before being output the accumulation is multiplied by the number of model timesteps in a sample period. This adjusts the accumulation for any intermittent sampling and is designed so that the time accumulation of a flux (e.g. kg s^{-1}) can easily be converted to a total (e.g. kg) by subsequently multiplying by the model timestep length during post processing.

6.35.3 Example of requesting output

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

```
&JULES_OUTPUT
  run_id = "jules_run001",

  output_dir = "./output",

  nprofiles = 2
/
```

(continues on next page)

(continued from previous page)

```
&JULES_OUTPUT_PROFILE
  profile_name = "month",

  output_main_run = T,

  output_period = -1,

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

&JULES_OUTPUT_PROFILE
  profile_name = "tstep",

  output_main_run = T,
  output_start    = "2000-01-01 00:00:00",
  output_end      = "2001-01-01 00:00:00",

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

Or using the alternative list syntax (see *Introduction to Fortran namelists*):

```
&JULES_OUTPUT
  # ... as above ...
/

&JULES_OUTPUT_PROFILE
  # ... as above ...

  nvars = 4,
  var(1) = "emis_gb",       var_name(1) = "Emissivity",    output_type(1) = 'M',
  var(2) = "ftl_gb",        var_name(2) = "SensibleHeat",  output_type(2) = 'M',
  var(3) = "snow_mass_gb",  var_name(3) = "SnowMass",      output_type(3) = 'M',
  var(4) = "tstar_gb",      var_name(4) = "Temp",          output_type(4) = 'M'
/

&JULES_OUTPUT_PROFILE
  # ... as above ...

  nvars = 4,
  var(1) = "emis",       output_type(1) = 'S',
  var(2) = "ftl",        output_type(2) = 'S',
  var(3) = "snow_mass",  output_type(3) = 'S',
  var(4) = "tstar",      output_type(4) = 'S'
/
```

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

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

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

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

6.36 oasis_rivers.nml

This file contains a single namelists called `OASIS_RIVERS`, which indicates how the Rivers-only executable couples via OASIS to other models (currently LFRIC and NEMO). This namelist is only used when running the river standalone program in coupled mode, i.e., compiling with the parameters RIVERS_ONLY and RIVER_CPL

Note: This namelist is only actually used when running the Rivers-only executable (compilation flag *RIVERS_ONLY*) in coupled mode (compilation flag *RIVER_CPL*)

6.36.1 OASIS_RIVERS namelist members

`OASIS_RIVERS::`**`np_receive`**

Type
integer

Permitted
2

Default
imdi

The number of fields that are received from other models via OASIS coupling.

`OASIS_RIVERS::`**`np_send`**

Type
integer

Permitted
0,1

Default
imdi

The number of fields that are sent to other models via OASIS coupling.

`OASIS_RIVERS::`**`cpl_freq`**

Type
integer

Permitted
1:

Default
imdi

The river coupling frequency in seconds.

Note: The river coupling frequency must be a multiple of the river executable time step, and of the time steps of the models to which it is coupled.

OASIS_RIVERS::**send_fields**

Type
character(:)

Permitted
'rflow_outflow'

Default
''

List of fields to be sent via coupling from the river executable to other models. Names are case sensitive

Note: The only field that can be sent via coupling is the total river runoff (*rflow_outflow*).

OASIS_RIVERS::**receive_fields**

Type
character(:)

Permitted
'sub_surf_roff_rp', 'surf_roff_rp'

Default
''

List of fields to be received by the river executable via coupling from other models. Names are case sensitive

Note: Coupled receive fields are used to substitute driving data read from file using the namelist *JULES_DRIVE* by the same fields generated by a driving model running in parallel to the river executable. The only fields that can be received via coupling are the surface runoff (*surf_roff_rp*) and the sub-surface runoff (*sub_surf_roff_rp*).

OASIS_RIVERS::**riv_number_file**

Type
character

Default
''

Ancillary file containing the river number. This information is necessary when sending via coupling the total runoff (*send_fields* = 'rflow_outflow'), so that rivers discharge in the right ocean grid point. The ancillary file identifies the river in which the river outflow on each grid point will discharge, so the total runoff for each river is calculated as the sum of the river outflow corresponding to that river.

6.36.2 Example of coupling request

In this example, the user has requested receiving the surface and sub-surface runoffs, and sending the total river runoff via coupling. The coupling exchanges take place every hour.

```
&OASIS_RIVERS
  cpl_freq = 3600,

  np_receive = 2,

  np_send = 1,
```

(continues on next page)

(continued from previous page)

```
  receive_fields = 'sub_surf_roff_rp','surf_roff_rp',

  riv_number_file = '$RIV_NUMBER_ANCILLARY/river_number_um.nc'

  send_fields = 'rflow_outflow',

/
```

CHAPTER

SEVEN

JULES EXAMPLES

If you are using a version of JULES since vn5.0, please see the list of standard jobs in the table given here or the ‘Example configurations’ row of the table here.

For earlier versions, please see chapter 8 of the appropriate JULES User Guide.

CHAPTER

EIGHT

ASPECTS OF THE CODE

8.1 I/O framework

JULES version 3.1 saw a complete rewrite of the I/O code to use a more modular and flexible structure. This section attempts to give a brief description of the low-level I/O framework, and explains how to make some commonly required changes.

Warning: This section requires a good knowledge of Fortran.

8.1.1 Overview

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

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

Dimension

A file has one or more dimensions. Each regular dimension has a name and a size.

One dimension is special, and is referred to as the record dimension. It has a name but has no defined size. A typical use of the record dimension is to represent time.

Variable

A file has one or more variables. The size of each variable is defined using the dimensions previously defined in the file. Each variable can opt to use the record dimension or not - if a variable uses the record dimension it must be the last dimension that the variable has.

Record

A record is the collection of all variables at a certain value of the record dimension. The figure *Records in a file* gives an example of this:

In the figure, each variable has dimensions x, y and n, where n is the record dimension. Each green box represents the (2D plane of) values of a variable for a certain value of n. A record is then the collection of all variables at a certain value of n.

A good analogy is the lines in an ASCII file, where each column represents a variable and each line is a record (in fact, this is a generalisation to multiple dimensions of that exact concept).

Files keep track of the record they are currently pointing at (it is the responsibility of the file-type drivers to do this in the way that best suits the file format they implement). When a file receives a read or write request for a particular variable, the values are read from or written to the current record.

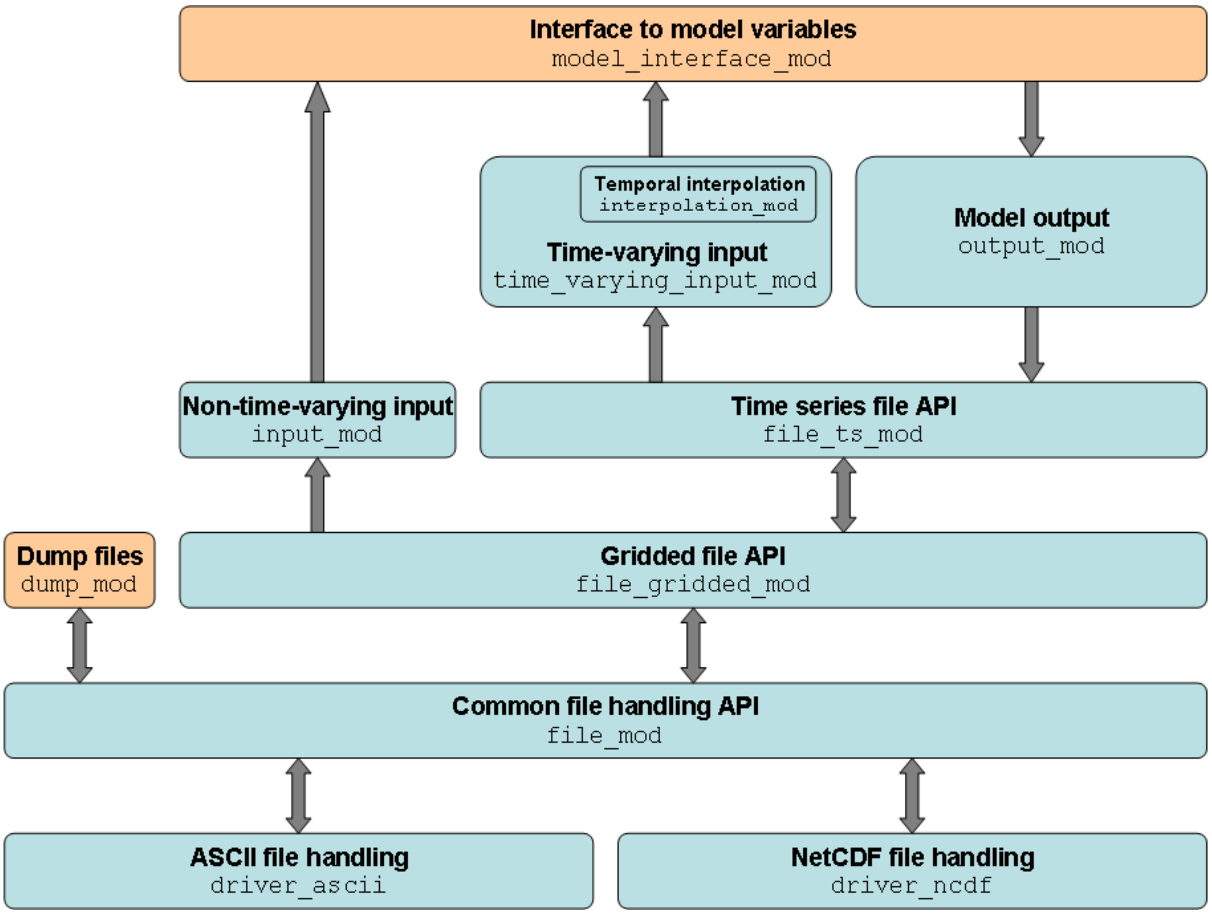

Fig. 1: Modular structure of the JULES I/O code

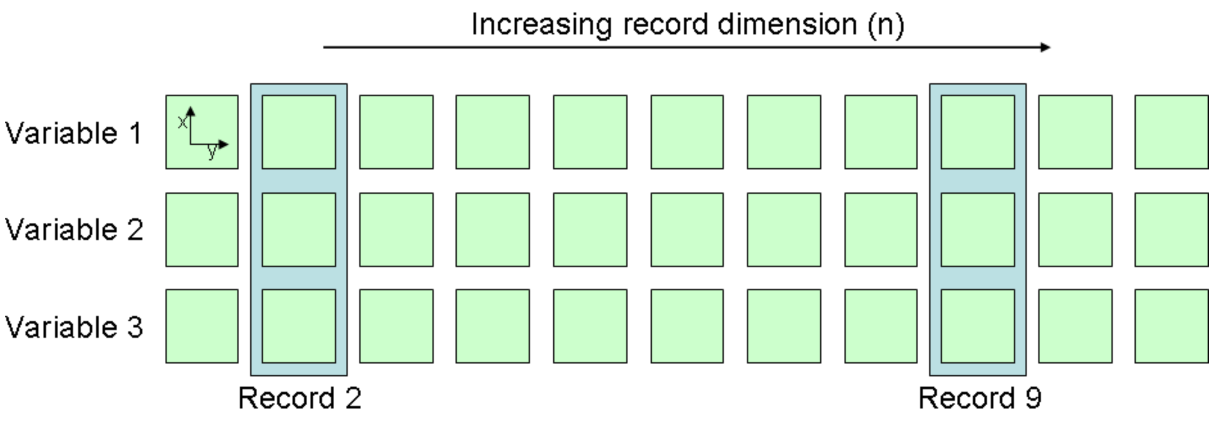

Fig. 2: Records in a file

The record abstraction also allows two useful operations - seek and advance. When a file receives an instruction to seek to a particular record, it sets its internal pointer so that read/write requests access the given record (a use of this within JULES is looping the input files round spin-up cycles). An advance instruction just moves the internal pointer on to the next record.

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

The gridded file API then imposes the concept of reading and writing cubes of gridded data (i.e. x and y dimensions for the grid, plus zero or more 'levels' dimensions) on top of the common file handling API. The underlying files may have a 1D or 2D grid (see *Input files for JULES*), and this layer handles the grid dimensions transparently. It is this layer that handles the extraction of a subgrid from a larger grid (see `file_gridded_read_var` and `file_gridded_write_var`).

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

The input and output layers interact with the model via an interface provided by `model_interface_mod`. `model_interface_mod` allows the input and output layers to read values from and write values to the internal model variables. This is discussed in more detail in *Implementing new variables for input and output*. The input and output layers use the time series file API to read from and write to file.

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

8.1.2 Implementing new variables for input and output

The only I/O code that needs to be modified to add new variables for input and output is in `model_interface_mod` (the routines in `src/io/model_interface`). All interaction between the I/O code and the model happens in this module (apart from reading and writing dump files).

Before adding any code to `model_interface_mod`, the variable the user wishes to make available for input and/or output must be accessible to `model_interface_mod`. This is usually accomplished by placing the variable in a module and importing the module into `model_interface_mod` where required, e.g.:

```fortran
! Declare the variable in a module
MODULE my_module

  REAL, ALLOCATABLE :: my_var(:)

  ! ...
END MODULE my_module


! ... Later, in model_interface_mod
USE my_module, ONLY : my_var
```

`model_interface_mod` contains several routines:

- Two routines that populate and extract data from the relevant model variables. These are `populate_var` and `extract_var` respectively.
- Routines that provide various pieces of information (e.g. string identifiers, number and size of 'levels' dimensions) about the available variables to the input and output layers. Internally, a metadata array that contains information about the available variables is used to implement these 'information providing' routines.

In most cases, the following edits will be sufficient to add a variable for input and/or output:

`model_interface_mod.F90`

Note: Required for both input and output variables.

Increment the constant `N_VARS`. This `PARAMETER` indicates how many elements are in the metadata array. If you forget to do this, the module will fail to compile.

`populate_var.inc`

Note: Required for input variables only.

`populate_var` takes a variable identifier and a cube of data on the full model grid, and populates the associated model variable using that data. This is done using a `SELECT` statement, to which a case must be added for the new variable.

`extract_var.inc`

Note: Required for output variables only.

`extract_var` takes a variable identifier, extracts the values from the associated model variable, and returns those values as a cube of data on the full model grid. This is done using a `SELECT` statement, to which a case must be added for the new variable.

`variable_metadata.inc`

Note: Required for both input and output variables.

This file contains the `DATA` definition for the variable metadata array. The metadata array contains objects of the derived type `var_metadata`, which is defined in `model_interface_mod.F90`. A typical entry in this array will look something like:

```
!-----------------------------------------------------------------------------
! Metadata for latitude
!-----------------------------------------------------------------------------
DATA metadata(1) / var_metadata(                                            &
! String identifier
    'latitude',                                                             &
! Variable type
    VAR_TYPE_SURFACE,                                                       &
! Long name
    "Gridbox latitude",                                                     &
! Units
    "degrees"                                                               &
  ) /
```

This allows us to define all the static information about a variable in one place:

String identifier
: This is the name used to identify the variable in namelists (as seen elsewhere in the User Guide)

Variable type
: This indicates the number and size of the 'levels' dimensions for the variable. For a full list of types see the file `get_var_levs_dims.inc`; some of the available types are:

Type	Number and size of 'levels' dimension(s)
VAR_TYPE_SURFACE	No levels dimension
VAR_TYPE_PFT	Single levels dimension of size *npft*
VAR_TYPE_NVG	Single levels dimension of size *nnvg*
VAR_TYPE_TYPE	Single levels dimension of size ntype (*npft* + *nnvg*)
VAR_TYPE_TILE	Single levels dimension of size nsurft (1 if *l_aggregate* = TRUE, ntype otherwise)
VAR_TYPE_SOIL	Single levels dimension of size *sm_levels*
VAR_TYPE_SCPOOL	Single levels dimension of size dim_cs1 (number of soil carbon pools, i.e. 4 if *l_triffid* = TRUE, 1 otherwise)
VAR_TYPE_SNOW	Two levels dimensions: the first of size nsurft and the second of size *nsmax*

Adding a new type is a relatively simple procedure:

1. A new PARAMETER must be added for the type in model_interface_mod.F90
2. A new CASE must be added to the SELECT statement in get_var_levs_dims.inc that correctly returns the number, names and sizes of the levels dimensions.

Long name
This is the name used in the long_name attribute for the variable in output files.

Units
This is the units given in the units attribute for the variable in output files.

map_from_land and map_to_land are provided as utilities for use with variables that are defined on land points only. tiles_to_gbm is used to provide gridbox mean diagnostics for model variables that have one value per surface tile.

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

8.1.3 Implementing a new file format

To understand how to implement a new file format, it first helps to understand how the common file handling layer works under the hood.

Each of the routines in file_mod (see files in src/io/file_handling/core) takes a file_handle as its first argument. The file_handle is a Fortran derived type that contains a flag indicating the format of the file it represents, and each of the routines in file_mod contains a SELECT statement that defers to the correct implementation of the routine based on that flag.

file_handles are created in file_open. Each file format implementation defines a list of recognised file extensions, and the appropriate file opening routine is deferred to by comparing the extension of the given file name to the recognised extensions for each file format.

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

Implementations of these routines for ASCII and NetCDF file formats are given in driver_ascii (see src/io/file_handling/core/drivers/ascii) and driver_ncdf (see src/io/file_handling/core/drivers/ncdf) respectively. These should be used as examples of how to implement a file format.

These two file formats suffer from opposite problems when implementing the concepts of dimensions, variables and records. For NetCDF, the concepts of dimensions and variables already exist, but the idea of a record has to be imposed. For ASCII, the concept of a record is a natural fit (think lines in a file), but the concepts of dimensions and variables have to be imposed. Between them, these implementations should provide sufficient examples of how to implement a new file format.

8.2 Known limitations of the code

8.2.1 Limit to longest possible run

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

8.2.2 Spin-up over short periods

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

8.2.3 Upgrade macros for the `JULES_VEGETATION_PROPS` namelist

The *JULES_VEGETATION_PROPS* namelist was added to the JULES source at vn5.7, but the upgrade macro to add this namelist to JULES Rose apps was not added until vn6.1. This means that when `rose app-upgrade` is used to upgrade a JULES app to versions vn5.7 through vn6.0, the *JULES_VEGETATION_PROPS* namelist will neither be added to the app and nor be described by the corresponding `rose-meta`. This namelist is needed only if *photo_acclim_model* is set to 1, in which case the user must manually edit their JULES app and `rose-meta` to include the relevant information. For this reason, we recommend using this science option only with JULES vn6.1 or later.

CHAPTER

NINE

JULES OUTPUT VARIABLES

Variables that are available for output from JULES are listed in this section, separated according to the broad area of science. If a variable cannot be found, users should also check in related sections.

Note: Most variables are output on the full model grid (even for variables defined on land points only).

Most river variables are output on the river routing model grid, which is a 1D grid defined on the valid river routing points only. These are indicated by names ending in `_rp` and have a single spatial dimension of size `np_rivers`. A regridded version of some river variables can also be output on the full model grid."

Any points on the grid for which a variable is not defined with be filled with a missing data value.

Any variables also available on soil tiles are indicated next to their non-soil-tiled analogues, e.g. `var[_soilt]`, in which case they have an additional dimension of size `nsoilt`.

All variables include a land point dimension, unless specified otherwise.

The sizes of dimensions are indicated in the tables below using links to other sections of this documentation, wherever possible. Other sizes are discussed in the table below.

Name	Description
`ch4layer`	Number of soil methane layers. Equals *`sm_levels`* if *`l_ch4_tlayered`* = TRUE, otherwise = 1.
`cslayer`	Number of layers for soil carbon and nitrogen. With the single-pool soil model (*`soil_bgc_model`* = 1), `cslayer` = 1. With 4-pool model (*`soil_bgc_model`* = 2), if *`l_layeredc`* = TRUE `cslayer` = *`sm_levels`*, otherwise `cslayer` = 1. With the ECOSSE model (*`soil_bgc_model`* = 3), `cslayer` = *`dim_cslayer`*.
`cspool`	Number of soil carbon pools. =1 with the single-pool soil model (*`soil_bgc_model`* = 1). =4 with the 4-pool or ECOSSE models (*`soil_bgc_model`* = 2 or 3).
`land+sea`	Variable is available on all points (land and sea).
`ncpft`	Number of crop plant functional types - see *`ncpft`*.
`npft`	Number of plant functional types - see *`npft`*.
`ns_deep`	The number of levels in the thermal-only bedrock - see *`ns_deep`*.
`nsmax`	Maximum-allowed number of snow layers - see *`nsmax`*.
`nsoilt`	Number of soil tiles. `nsurft` if *`l_tile_soil`* = TRUE, otherwise 1.
`nsurft`	Number of surface tiles. 1 if *`l_aggregate`* = TRUE, otherwise `ntype`.
`ntype`	Number of surface types, = *`npft`* + *`nnvg`*
`sm_levels`	Number of soil layers (for soil moisture) - see *`sm_levels`*.

9.1 Meteorology

Unlesss stated otherwise these variables have values at both land and sea points.

Name	Description	Dimensions
`precip`	Gridbox precipitation rate (kg m^{-2} s^{-1}).	
`rainfall`	Gridbox rainfall rate (kg m^{-2} s^{-1}).	
`snowfall`	Gridbox snowfall rate (kg m^{-2} s^{-1}).	
`con_rain`	Gridbox convective rainfall (kg m^{-2} s^{-1}).	
`con_snow`	Gridbox convective snowfall (kg m^{-2} s^{-1}).	
`ls_rain`	Gridbox large-scale rainfall (kg m^{-2} s^{-1}).	
`ls_snow`	Gridbox large-scale snowfall (kg m^{-2} s^{-1}).	
`pstar`	Gridbox surface pressure (Pa).	
`q1p5m_gb`	Gridbox specific humidity at 1.5m height (kg kg^{-1}).	
`qw1`	Gridbox specific humidity (total water content) (kg kg^{-1}).	
`q1p5m`	Tile specific humidity at 1.5m over land tiles (kg kg^{-1}).	land,nsurft
`lw_down`	Gridbox surface downward LW radiation (W m^{-2}).	
`sw_down`	Gridbox surface downward SW radiation (W m^{-2}).	
`t1p5m_gb`	Gridbox temperature at 1.5m height (K).	
`t1p5m`	Tile temperature at 1.5m over land tiles (K).	land,nsurft
`tl1`	Gridbox ice/liquid water 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}).	

9.2 Radiation

Name	Description	Dimensions
Albedos and emissivities		
`albedo_land`	Gridbox albedo (as used to calculate net shortwave radiation) (-).	
`alb_tile_1`	Tile land albedo, waveband 1 (direct beam visible).	nsurft
`alb_tile_2`	Tile land albedo, waveband 2 (diffuse visible).	nsurft
`alb_tile_3`	Tile land albedo, waveband 3 (direct beam NIR).	nsurft
`alb_tile_4`	Tile land albedo, waveband 4 (diffuse NIR).	nsurft
`land_albedo_1`	Gridbox band 1 albedo (direct beam visible).	land+sea
`land_albedo_2`	Gridbox band 2 albedo (diffuse visible).	land+sea
`land_albedo_3`	Gridbox band 3 albedo (direct beam NIR).	land+sea
`land_albedo_4`	Gridbox band 4 albedo (diffuse NIR).	land+sea
`emis_gb`	Gridbox emissivity.	
`emis`	Tile emissivity.	nsurft
Radiation fluxes		
`apar`	PFT absorbed photosynthetically active radiation (W m^{-2}).	npft
`apar_gb`	Gridbox absorbed photosynthetically active radiation (W m^{-2}).	
`lw_down_surft`	Tile downwelling longwave radiation (W m^{-2}).	nsurft
`lw_up_surft`	Tile upwelling longwave radiation (W m^{-2}).	nsurft
`lw_net`	Gridbox surface net LW radiation (W m^{-2}).	
`lw_up`	Gridbox surface upward LW radiation (W m^{-2}).	
`rad_net`	Surface net radiation (W m^{-2}).	
`rad_net_tile`	Tile surface net radiation (W m^{-2}).	nsurft
`sw_net`	Gribox net shortwave radiation at the surface (W m^{-2}).	
`sw_net_surft`	Tile net shortwave radiation (W m^{-2}).	nsurft
Other radiation variables		
`cosz`	Cosine of the zenith angle (-).	land+sea
`diff_frac`	Gridbox fraction of radiation that is diffuse (-).	land+sea
`fapar`	PFT fraction of absorbed photosynthetically active radiation (-).	npft
`NDVI_land`	Gridbox NDVI (using sum of direct and diffuse for (NIR-VIS)/(NIR+VIS)).	
`trad`	Gridbox effective radiative temperature (K).	

9.3 Energy and momentum fluxes, and surface temperatures

Name	Description	Dimensions
`ftl`	Tile surface sensible heat flux for land tiles (W m^{-2}).	nsurft
`ftl_gb`	Gridbox surface sensible heat flux (W m^{-2}).	land+sea
`le`	Tile surface latent heat flux for land tiles (W m^{-2}).	nsurft
`latent_heat`	Gridbox surface latent heat flux (W m^{-2}).	land+sea
`surf_ht_flux`	Downward heat flux for each tile (W m^{-2}).	nsurft
`surf_ht_store`	C*(dT/dt) for each tile (W m^{-2}).	nsurft
`surf_ht_flux_gb`	Gridbox net downward heat flux at surface over land and sea-ice fraction of gridbox (W m^{-2}).	land+sea
`anthrop_heat`	Anthropogenic heat flux for each tile (W m^{-2}).	nsurft
`hf_snow_melt`	Gridbox snowmelt heat flux (W m^{-2}).	
`snomlt_surf_htf`	Gridbox heat flux used for surface melting of snow (W m^{-2}).	land+sea
`snomlt_sub_htf`	Gridbox sub-canopy snowmelt heat flux (W m^{-2}).	
`tstar_gb`	Gridbox surface temperature (K).	land+sea
`tstar`	Tile surface temperature (K).	nsurft
`tsurf_elev_surft`	Tile temperature of elevated subsurface tiles (K).	nsurft
`tau`	Tile surface wind stress for land tiles (N m^{-2}).	nsurft
`taux1`	Gridbox westerly component of surface wind stress (N m^{-2}).	land+sea
`tauy1`	Gridbox southerly component of surface wind stress (N m^{-2}).	land+sea
`tauy_gb`	Gridbox scalar magnitude of surface wind stress (N m^{-2}).	land+sea
`z0`	Tile surface roughness (m).	nsurft

9.4 Soil moisture and temperature, and soil characteristics

Name	Description	Dimensions
Soil moisture		
`smcl[_soilt]`	Moisture content of each soil layer (kg m^{-2}).	sm_levels
`soil_wet`	Total moisture content of each soil layer, as fraction of saturation (-).	sm_levels
`sthu[_soilt]`	Unfrozen moisture content of each soil layer as a fraction of saturation (-).	sm_levels
`sthu_irr[_soilt]`	Unfrozen moisture content of each soil layer as a fraction of saturation in irrigated fraction (-) (only available if l_irrig_dmd = T).	sm_levels
`sthf[_soilt]`	Frozen moisture content of each soil layer as a fraction of saturation (-).	sm_levels
`smc_tot`	Gridbox total soil moisture in column (kg m^{-2}).	
`swet_liq_tot`	Gridbox unfrozen soil moisture as fraction of saturation (-).	
`swet_tot`	Gridbox soil moisture as fraction of saturation (-).	
`sthzw[_soilt]`	Soil wetness in the deep LSH/TOPMODEL layer (-).	
`zw[_soilt]`	Gridbox mean depth to water table (m).	
Soil temperature		
`t_soil[_soilt]`	Sub-surface temperature of each layer (K).	sm_levels
`tsoil_deep`	Temperature of each bedrock layer (K). Only available when *l_bedrock* = TRUE.	ns_deep
`depth_frozen`	Gridbox depth of frozen ground at surface defined from soil temperature (m).	
`depth_frozen_sthf`	Gridbox depth of frozen ground at surface defined from soil moisture (m). Recommended over `depth_frozen` except where the soil is very dry.	
`depth_unfrozen`	Gridbox depth of unfrozen ground at surface defined from soil temperature (m).	
`depth_unfrozen_sthf`	Gridbox depth of unfrozen ground at surface defined from soil moisture (m). Recommended over `depth_unfrozen` except where the soil is very dry.	
Soil characteristics		
`b[_soilt]`	Brooks-Corey exponent for each soil layer (-).	sm_levels
`hcap[_soilt]`	Dry soil heat capacity (J K^{-1} m^{-3}) for each soil layer.	sm_levels
`hcon[_soilt]`	Dry soil thermal conductivity (W m^{-1} K^{-1}) for each soil layer.	sm_levels
`satcon[_soilt]`	Saturated hydraulic conductivity (kg m^{-2} s^{-1}) for each soil layer.	sm_levels
`sathh[_soilt]`	Saturated soil water pressure (m) for each soil layer.	sm_levels
`sm_crit[_soilt]`	Volumetric moisture content at critical point for each soil layer (-), as given in *JULES_SOIL_PROPS*.	sm_levels
`sm_sat[_soilt]`	Volumetric moisture content at saturation for each soil layer (-).	sm_levels
`sm_wilt[_soilt]`	Volumetric moisture content at wilting point for each soil layer (-), as given in *JULES_SOIL_PROPS*.	sm_levels

9.5 Hydrology

Name	Description	Dimensions
Canopy hydrology		
`canopy_gb`	Gridbox canopy water content (kg m^{-2}).	
`canopy`	Tile surface/canopy water for snow-free land tiles (kg m^{-2}).	nsurft
`catch`	Tile surface/canopy water capacity of snow-free land tiles (kg m^{-2}).	nsurft

continues on next page

Table 1 – continued from previous page

Name	Description	Dimensions
tfall	Gridbox throughfall (kg m^{-2} s^{-1}).	
Evaporation and sublimation		
fqw	Tile surface moisture flux for land tiles (kg m^{-2} s^{-1}).	nsurft
fqw_gb	Gridbox moisture flux from surface (kg m^{-2} s^{-1}).	land+sea
ecan	Tile evaporation from canopy/surface store for snow-free land tiles (kg m^{-2} s^{-1}).	nsurft
ecan_gb	Gridbox mean evaporation from canopy/surface store (kg m^{-2} s^{-1}).	land+sea
ei	Tile sublimation from lying snow for land tiles (kg m^{-2} s^{-1}).	nsurft
ei_gb	Gridbox sublimation from lying snow or sea-ice (kg m^{-2} s^{-1}).	land+sea
elake	Gridbox mean evaporation from lakes (kg m^{-2} s^{-1}).	
esoil	Tile surface evapotranspiration from soil moisture store for snow-free land tile (kg m^{-2} s^{-1}).	nsurft
esoil_gb	Gridbox surface evapotranspiration from soil moisture store (kg m^{-2} s^{-1}).	land+sea
et_stom	Tile transpiration (kg m^{-2} s^{-1}).	nsurft
et_stom_gb	Gridbox transpiration (kg m^{-2} s^{-1}).	land+sea
fao_et0	FAO Penman-Monteith evapotranspiration for reference crop (kg m^{-2} s^{-1})	
gc	Tile surface conductance to evaporation for land tiles (m s^{-1}).	nsurft
gs	Gridbox surface conductance to evaporation (m s^{-1}).	
ext[_soilt]	Extraction of water from each soil layer (kg m^{-2} s^{-1}).	sm_levels
fsmc_gb	Gridbox soil moisture availability factor (beta) (-).	
fsmc	PFT soil moisture availability factor (-).	npft
smc_avail_top	Gridbox available moisture in surface layer of depth given by *zsmc* (kg m^{-2}). Calculated using sm_wilt from *JULES_SOIL_PROPS*.	
smc_avail_tot	Gridbox available moisture in soil column (kg m^{-2}). Calculated using sm_wilt from *JULES_SOIL_PROPS*.	
Runoff		
runoff	Gridbox runoff rate (kg m^{-2} s^{-1}).	
sub_surf_roff	Gridbox sub-surface runoff (kg m^{-2} s^{-1}).	
surf_roff	Gridbox surface runoff (kg m^{-2} s^{-1}).	
sat_excess_roff[_soilt]	Gridbox saturation excess runoff rate (kg m^{-2} s^{-1}).	
drain[_soilt]	Gridbox drainage at bottom of soil column (kg m^{-2} s^{-1}).	
qbase[_soilt]	Gridbox baseflow (lateral subsurface runoff) (kg m^{-2} s^{-1}), i.e. the sum of surface and subsurface lateral flows from all soil layers (inc. deep LSH/TOPMODEL layer). Only available if *l_top* = TRUE.	
qbase_zw[_soilt]	Gridbox baseflow (lateral subsurface runoff) from deep LSH/TOPMODEL layer (kg m^{-2} s^{-1}). Only available if *l_top* = TRUE.	
Other hydrological variables		
fsat[_soilt]	Gridbox surface saturated fraction (-). The fraction of grid cell where the water table is above the land surface. Only available if *l_top* = TRUE.	
fwetl[_soilt]	Gridbox wetland fraction at end of model timestep (-). The fraction of grid cell where the water table is above the land surface, but water is not flowing (stagnant) (fwetl<=fsat). Only available if *l_top* = TRUE.	

9.6 Rivers

Name	Description	Dimensions
Output on the river routing model grid		
rflow_rp	River routing gridbox river flow rate (kg m^{-2} s^{-1}). Only available if l_rivers = TRUE.	np_rivers
rrun_rp	River routing gridbox runoff rate received by river routing routine (kg m^{-2} s^{-1}). Only available if l_rivers = TRUE.	np_rivers
rrun_surf_rp	River routing gridbox surface runoff rate received by river routing routine (kg m^{-2} s^{-1}). Only available if l_rivers = TRUE.	np_rivers
rrun_sub_surf_rp	River routing gridbox sub-surface runoff rate received by river routing routine (kg m^{-2} s^{-1}). Only available if l_rivers = TRUE.	np_rivers
rfm_surfstore_rp	Surface storage on river points (m^3). Only available if l_rivers = TRUE and i_river_vn = 2.	np_rivers
rfm_substore_rp	Sub-surface storage on river points (m^3). Only available if l_rivers = TRUE and i_river_vn = 2.	np_rivers
rfm_flowin_rp	Surface inflow on river points (m^3 s^{-1}). Only available if l_rivers = TRUE and i_river_vn = 2.	np_rivers
rfm_bflowin_rp	Sub-surface inflow on river points (m^3 s^{-1}). Only available if l_rivers = TRUE and i_river_vn = 2.	np_rivers
rivers_sto_rp	River routing gridbox river storage (kg) Only available if l_rivers = TRUE and i_river_vn = 3.	np_rivers
frac_fplain_rp	Overbank inundation area as a fraction of river routing gridcell area. Only available if l_riv_overbank = TRUE.	np_rivers
Output regridded to the JULES model grid		
rflow	Gridbox river flow rate (kg m^{-2} s^{-1}). Only available if l_rivers = TRUE.	
rrun	Gridbox runoff rate received by river routing routine (kg m^{-2} s^{-1}). Only available if l_rivers = TRUE	
frac_fplain_lp	Overbank inundation area as a fraction of gridcell area. Only available if l_riv_overbank = TRUE.	

9.7 Snow

Name	Description	Dimensions
Snow state		
snow_mass	Tile lying snow (total) (kg m^{-2}).	nsurft
snow_mass_gb	Gridbox snowmass (kg m^{-2}).	land+sea
snow_depth	Tile snow depth (m).	nsurft
snow_depth_gb	Gridbox depth of snow (m).	
snow_can	Tile snow on canopy (kg m^{-2}).	nsurft
snow_can_gb	Gridbox snow on canopy (kg m^{-2}).	
snow_ground	Tile snow on ground (snow_tile or snow_grnd depending on configuration) (kg m^{-2}).	nsurft
snow_grnd_gb	Gridbox average snow beneath canopy (snow_grnd) (kg m^{-2}).	
snow_grnd	Tile snow on ground below canopy (kg m^{-2}).	nsurft
snow_grnd_rho	Tile bulk density of snow on ground (kg m^{-3}).	nsurft
snow_frac	Gridbox snow-covered fraction of land points (-).	
snow_ice_tile	Tile total frozen mass in snow on ground (kg m^{-2}). Only available if nsmax > 0.	nsurft

continues on next page

Table 2 – continued from previous page

Name	Description	Dimensions
`snow_ice_gb`	Gridbox frozen water in snowpack (kg m^{-2}). Only available if *nsmax* > 0.	
`snow_liq_tile`	Tile total liquid mass in snow on ground (kg m^{-2}). Only available if *nsmax* > 0.	nsurft
`snow_liq_gb`	Gridbox liquid water in snowpack (kg m^{-2}). Only available if *nsmax* > 0.	
`nsnow`	Tile number of snow layers (-).	nsurft
`rgrain`	Tile snow surface grain size (µm).	nsurft
Snow layer variables		
`snow_ds`	Depth of each snow layer for each tile (m). Only available if *nsmax* > 0.	nsurft,nsmax
`snow_ice`	Mass of ice in each snow layer for each tile (kg m^{-2}). Only available if *nsmax* > 0.	nsurft,nsmax
`snow_liq`	Mass of liquid water in each snow layer for each tile (kg m^{-2}). Only available if *nsmax* > 0.	nsurft,nsmax
`tsnow`	Temperature of each snow layer (K). Only available if *nsmax* > 0.	nsurft,nsmax
`rgrainl`	Grain size in snow layers for each tile (µm). Only available if *nsmax* > 0.	nsurft,nsmax
Snow fluxes and rates of change		
`snow_melt`	Tile snow melt rate (melt_tile) (kg m^{-2} s^{-1}).	nsurft
`snow_melt_gb`	Gridbox rate of snowmelt (kg m^{-2} s^{-1}).	
`snow_can_melt`	Tile melt of snow on canopy (kg m^{-2} s^{-1}).	nsurft
`snice_freez_surft`	Tile internal refreezing rate in snowpack (kg m^{-2} s^{-1}).	nsurft
`snice_m_surft`	Tile total internal melt rate of snowpack (kg m^{-2} s^{-1}).	nsurft
`snice_runoff_surft`	Tile net rate of liquid leaving snowpack on tiles (kg m^{-2} s^{-1}).	nsurft
`snice_sicerate_surft`	Tile rate of change of solid mass in snowpack (kg m^{-2} s^{-1}).	nsurft
`snice_sliqrate_surft`	Tile rate of change of liquid in snowpack (kg m^{-2} s^{-1}).	nsurft
`snice_smb_surft`	Tile rate of change of snowpack mass (kg m^{-2} s^{-1}).	nsurft
`snow_soil_htf`	Tile downward heat flux after snowpack to subsurface" (W m^{-2}).	nsurft

9.8 Vegetation carbon and related fluxes

Name	Description	Dimensions
`c_veg`	**PFT total carbon content of the vegetation at end of model timestep (kg C m^{-2}).** (including leaf, wood and root carbon, both above and below ground)	npft
`cv`	Gridbox mean vegetation carbon at end of model timestep (kg m^{-2}).	
`leafC`	PFT carbon in leaf biomass (kg m^{-2}).	npft
`rootC`	PFT carbon in root biomass (kg m^{-2}).	npft
`woodC`	PFT carbon in woody biomass (kg m^{-2}).	npft
`frac_agr`	Fractional area of agricultural land in each gridbox. If *l_trif_crop* is TRUE, frac_agr is the fractional area of crop land in each gridbox.	
`frac_agr_prev`	Fractional area of agricultural land at the previous timestep.	
`frac_past`	Fractional area of pasture land in each gridbox.	
`plantNumDensity`	Number density of plants (m^{-2}).	npft,nmasst
Fractional cover, leaf area and turnover, and canopy height		
`frac`	Fractional cover of each surface type.	ntype
`lai`	PFT leaf area index (-).	npft

continues on next page

Table 3 – continued from previous page

Name	Description	Dimensions
`lai_gb`	Gridbox leaf area index (-).	
`lai_bal`	PFT balanced leaf area index in sf_stom (-).	npft
`lai_phen`	PFT leaf area index after phenology (-).	npft
`canht`	PFT canopy height (m).	npft
`g_leaf`	PFT leaf turnover rate ([360days]$^{-1}$).	npft
`g_leaf_day`	PFT mean leaf turnover rate for input to PHENOL ([360days]$^{-1}$).	npft
`g_leaf_dr_out`	PFT mean leaf turnover rate for driving TRIFFID ([360days]$^{-1}$).	npft
`g_leaf_phen`	PFT mean leaf turnover rate over phenology period([360days]$^{-1}$).	npft
GPP, NPP, respiration		
`gpp`	PFT gross primary productivity of biomass expressed as carbon (kg C m^{-2} s^{-1}).	npft
`gpp_gb`	Gridbox gross primary productivity of biomass expressed as carbon (kg C m^{-2} s^{-1}).	
`npp`	PFT net primary productivity of biomass expressed as carbon prior to nitrogen limitation (kg C m^{-2} s^{-1}).	npft
`npp_n_gb`	Gridbox net primary productivity of biomass expressed as carbon after nitrogen limitation (kg C m^{-2} (360days)$^{-1}$).	
`npp_n`	PFT net primary productivity of biomass expressed as carbon after nitrogen limitation (kg C m^{-2} (360days)$^{-1}$).	npft
`npp_dr_out`	PFT mean NPP of biomass expressed as carbon for driving TRIFFID (kg C m^{-2} (360days)$^{-1}$).	npft
`resp_p`	PFT plant respiration carbon flux (kg m^{-2} s^{-1}).	npft
`resp_p_gb`	Gridbox plant respiration carbon flux (kg m^{-2} s^{-1}).	
`resp_w_dr_out`	PFT mean wood respiration carbon flux for driving TRIFFID (kg m^{-2} (360days)$^{-1}$).	npft
`resp_l`	PFT leaf respiration carbon flux (kg m^{-2} s^{-1}).	npft
`resp_r`	PFT root respiration carbon flux (kg m^{-2} s^{-1}).	npft
`resp_w`	PFT wood respiration carbon flux (kg m^{-2} s^{-1}).	npft
Litter carbon fluxes		
`lit_c`	PFT carbon litter (kg m^{-2} (360days)$^{-1}$).	npft
`lit_c_mean`	Gridbox mean carbon litter (kg m^{-2} (360days)$^{-1}$).	
`lit_c_ag`	PFT carbon litter from LU/agriculture (kg C m^{-2} (360days)$^{-1}$).	npft
`lit_c_orig`	PFT carbon litter including LU (kg C m^{-2} (360days)$^{-1}$).	npft
`leaf_litC`	PFT litter carbon due to leaf turnover (kg m^{-2})(360days)$^{-1}$).	npft
`root_litC`	PFT litter carbon due to root turnover (kg m^{-2})(360days)$^{-1}$).	npft
`wood_litC`	PFT litter carbon due to wood turnover (kg m^{-2})(360days)$^{-1}$).	npft
`plant_input_c_gb`	Gridbox input of C to the soil by plant litterfall (kg m^{-2} s^{-1}). Only available with the ECOSSE soil model (`soil_bgc_model` = 3).	
Other carbon fluxes		
`exudates`	PFT exudates - excess carbon not assimilable into plant due lack of nitrogen availability (kg C m^{-2} (360days)$^{-1}$).	npft
`exudates_gb`	Gridbox exudates: excess carbon not assimilable into plant due lack of nitrogen (kg m^{-2} (360days)$^{-1}$).	
`pc_s`	PFT net carbon available for spreading in TRIFFID (kg m^{-2} (360 days)$^{-1}$).	npft
Harvest, wood products and land use		
`root_abandon`	PFT carbon flux from roots abandoned during landuse change to soil (kg C m^{-2} (360days)$^{-1}$).	npft
`root_abandon_gb`	Carbon from roots abandoned during landuse change to soil kg C m^{-2} (360days)$^{-1}$).	
`harvest`	Flux of carbon to product pools due to harvest (kg C m^{-2} (360days)$^{-1}$).	npft
`harvest_gb`	Gridbox flux of carbon to product pools due to harvest (kg C m^{-2} (360days)$^{-1}$).	

continues on next page

Table 3 – continued from previous page

Name	Description	Dimensions
harvest_biocrop	Flux of carbon to product pools due to biocrop harvest (kg C m^{-2} (360days)$^{-1}$).	npft
harvest_biocrop_gb	Gridbox flux of carbon to product pools due to biocrop harvest (kg C m^{-2} (360days)$^{-1}$).	
wood_prod_fast	Carbon content of the fast decay-rate wood product pool (kg m^{-2}).	
wood_prod_med	Carbon content of the medium decay-rate wood product pool (kg m^{-2}).	
wood_prod_slow	Carbon content of the slow decay-rate wood product pool (kg m^{-2}).	
WP_fast_in	Carbon flux from vegetation to the fast decay-rate wood product pool (kg m^{-2} [360days]$^{-1}$).	
WP_med_in	Carbon flux from vegetation to the medium decay-rate wood product pool (kg m^{-2} [360days]$^{-1}$).	
WP_slow_in	Carbon flux from vegetation to the slow decay-rate wood product pool (kg m^{-2} [360days]$^{-1}$).	
WP_fast_out	Carbon flux from the fast decay-rate wood product pool to atmosphere (kg m^{-2} [360days]$^{-1}$).	
WP_med_out	Carbon flux from the medium decay-rate wood product pool to atmosphere (kg m^{-2} [360days]$^{-1}$).	
WP_slow_out	Carbon flux from the slow decay-rate wood product pool to atmosphere (kg m^{-2} [360days]$^{-1}$).	
Carbon conservation		
cnsrv_carbon_veg2	Error in land carbon conservation in veg2 routine (kg m-2)	
cnsrv_carbon_triffid	Error in land carbon conservation in triffid routine (kg m-2)	
cnsrv_veg_triffid	Error in vegetation carbon conservation in triffid routine (kg m-2)	
cnsrv_soil_triffid	Error in soil carbon conservation in triffid routine (kg m-2)	
cnsrv_prod_triffid	Error in wood product carbon conservation in triffid routine (kg m-2)	
Thermal acclimation of photosynthesis		
t_home_gb	Long-term home temperature for C3 photosynthesis (K). Only available if *photo_acclim_model* = 1 or 3.	
t_growth_gb	Short-term growth temperature for C3 photosynthesis (K). Only available if *photo_acclim_model* = 2 or 3.	

9.9 Vegetation nitrogen and related fluxes

Name	Description	Dimensions
n_veg	PFT plant nitrogen content N_LEAF+N_ROOT+N_WOOD from carbon equivalents (kg m^{-2}).	npft
n_veg_gb	Gridbox mean plant nitrogen content: n_leaf+n_root+n_wood from carbon equivalents (kg m^{-2}).	
n_leaf	PFT leaf nitrogen scaled by LAI in sf_stom (kg m^{-2}).	npft
n_root	PFT root nitrogen scaled by LAI_BAL in sf_stom (kg m^{-2}).	npft
n_stem	PFT stem nitrogen scaled by LAI in sf_stom; scaled by LAI_BAL if l_stem_resp_fix=T (kg m^{-2}).	npft
Nitrogen fluxes		
deposition_n	Nitrogen deposition (kg m^{-2} s^{-1}).	
n_demand	PFT total nitrogen demand (kg m^{-2} (360 days)$^{-1}$).	npft
n_demand_gb	Gridbox mean demand for nitrogen (kg m^{-2} (360days)$^{-1}$).	
n_fix	PFT fixed nitrogen (kg N m^{-2} (360days)$^{-1}$).	npft
n_fix_gb	Gridbox mean nitrogen fixed by plants (kg m^{-2} (360days)$^{-1}$).	
n_uptake	PFT nitrogen taken up by plants (kg m^{-2} (360 days)$^{-1}$).	npft
n_uptake_gb	Gridbox total nitrogen uptake by plants (kg N m^{-2} (360days)$^{-1}$). Only available if *soil_bgc_model* = 2 or 3.	

continues on next page

Table 4 – continued from previous page

Name	Description	Dimensions
n_demand_growth	PFT nitrogen demand for growth of existing plant biomass (kg m^{-2} (360 days)$^{-1}$).	npft
n_uptake_growth	PFT nitrogen taken up for growth of existing plant biomass (kg m^{-2} (360 days)$^{-1}$).	npft
n_demand_lit	PFT nitrogen demand of litter: nitrogen lost in leaf, wood and root biomass (kg m^{-2} (360 days)$^{-1}$).	npft
n_demand_spread	PFT nitrogen demand for spreading plants across gridbox (kg m^{-2} (360 days)$^{-1}$).	npft
n_fertiliser	Nitrogen addition from fertiliser (kg N m^{-2} (360days)$^{-1}$).	npft
Nitrogen fluxes in litter, harvest and land use		
lit_n_t	Gridbox mean total nitrogen litter (kg m^{-2} (360days)$^{-1}$). (kg m^{-2} s^{-1}).	
lit_n	PFT nitrogen litter (kg N m^{-2} (360days)$^{-1}$).	npft
leaf_litN	PFT litter nitrogen due to leaf turnover (kg m^{-2})(360days)$^{-1}$).	npft
lit_n_ag	PFT nitrogen loss due to LU/agriculture (kg N m^{-2} (360days)$^{-1}$).	npft
litterN	PFT local nitrogen litter production (kg N m^{-2} (360days)$^{-1}$).	npft
lit_n_orig	PFT nitrogen litter including LU (kg N m^{-2} (360days)$^{-1}$).	npft
root_litN	PFT nitrogen lost as litter due to root turnover (kg m^{-2})(360days)$^{-1}$).	npft
wood_litN	PFT litter nitrogen due to wood turnover (kg m^{-2})(360days)$^{-1}$).	npft
plant_input_n_gb	Gridbox input of N to the soil by plant litterfall (kg m^{-2} s^{-1}). Only available with the ECOSSE soil model (*soil_bgc_model* = 3).	
harvest_n	flux of nitrogen to atmosphere due to harvest (kg N m^{-2} (360days)$^{-1}$).	npft
harvest_n_gb	Gridbox flux of nitrogen to atmosphere due to harvest (kg N m^{-2} (360days)$^{-1}$).	
harvest_biocrop_n	flux of nitrogen to product pools due to biocrop harvest (kg N m^{-2} (360days)$^{-1}$).	npft
harvest_biocrop_n_gb	Gridbox flux of nitrogen to product pools due to biocrop harvest (kg N m^{-2} (360days)$^{-1}$).	
root_abandon_n	PFT nitrogen flux from roots abandoned during landuse change to soil (kg N m^{-2} (360days)$^{-1}$).	npft
root_abandon_n_gb	Nitrogen from roots abandoned during landuse change to soil kg N m^{-2} (360days)$^{-1}$).	
Nitrogen conservation		
cnsrv_nitrogen_triffd	Error in land nitrogen conservation in triffid routine (kg m-2)	
cnsrv_vegN_triffid	Error in vegetation nitrogen conservation in triffid routine (kg m-2)	
cnsrv_soilN_triffid	Error in soil nitrogen conservation in triffid routine (kg m-2)	
cnsrv_n_inorg_triffid	Error in inorganic nitrogen conservation in triffid routine (kg m-2)	

9.10 Soil carbon and related fluxes

Name	Description	Dimensions
cs[_soilt]	Carbon in each soil pool and each soil biogeochemistry layer (kg m^{-2}).	cspool,cslayer
cs_label	Labelled carbon in each soil pool and each soil biogeochemistry layer (kg m^{-2}). Only available if *l_label_frac_cs* = TRUE.	cspool,cslayer
cs_gb	Gridbox total soil carbon (kg m^{-2}).	

continues on next page

Table 5 – continued from previous page

Name	Description	Dimensions
cs_label_gb	Gridbox total labelled soil carbon (kg m^{-2}). Only available if *l_label_frac_cs* = TRUE.	
c_bio_gb	Gridbox soil carbon in biomass pool (kg m^{-2}). Only available if *soil_bgc_model* = 2 or 3.	
c_dpm_gb	Gridbox soil carbon in decomposable plant material pool (kg m^{-2}). Only available if *soil_bgc_model* = 2 or 3.	
c_hum_gb	Gridbox soil carbon in humus pool (kg m^{-2}). Only available if *soil_bgc_model* = 2 or 3.	
c_rpm_gb	Gridbox soil carbon in resistant plant material pool (kg m^{-2}). Only available if *soil_bgc_model* = 2 or 3.	
Soil carbon fluxes		
co2_soil_gb	Gridbox C in CO_2 flux from soil to atmosphere (kg m^{-2} s^{-1}). Only available with the ECOSSE soil model (*soil_bgc_model* = 3).	
resp_s	Respiration rate from each soil carbon pool each soil biogeochemistry layer (kg m^{-2} s^{-1}).	cspool,cslayer
resp_label_cs	Respiration rate from each labelled soil carbon pool each soil biogeochemistry layer (kg m^{-2} s^{-1}). Only available if *l_label_frac_cs* = TRUE.	cspool,cslayer
resp_s_gb	Gridbox total soil respiration carbon flux (kg m^{-2} s^{-1}).	
resp_label_cs_gb	Gridbox total labelled soil respiration carbon flux (kg m^{-2} s^{-1}). Only available if *l_label_frac_cs* = TRUE.	
resp_s_to_atmos_gb	Respired carbon from soil carbon emitted to atmosphere (kg m^{-2} s^{-1}).	
resp_s_dr_out	Gridbox mean soil respiration carbon flux for driving TRIFFID (kg m^{-2} (360days)$^{-1}$) This is the gross soil respiration; some of this carbon flux is from one soil carbon pool to another.	
Other soil carbon variables		
dpm_ratio	Gridbox DPM:RPM ratio of overall litter input (:).	
fsth	Soil moisture modifier of soil respiration rate (-).	sm_levels
ftemp	Temperature modifier of soil respiration rate (-).	sm_levels
fprf	Modifier of soil respiration rate due to vegetation cover (-).	
soil_CN	Soil C:N in each soil pool and each soil biogeochemistry layer	cspool,cslayer
soil_cn_gb	Gridbox total soil carbon : nitrogen ratio.	
Soil methane variables		
fch4_wetl	Gridbox scaled methane flux from wetland fraction using soil carbon as substrate if ch4_substrate=1, NPP as substrate if ch4_substrate=2, or soil respiration as substrate if ch4_substrate=3 (10^{-9} kg m^{-2} s^{-1}).	
fch4_wetl_cs[_soilt]	Gridbox methane flux from wetland fraction using soil carbon as substrate (kg m^{-2} s^{-1}).	
fch4_wetl_npp[_soilt]	Gridbox methane flux from wetland fraction using NPP as substrate (kg m^{-2} s^{-1}).	
fch4_wetl_resps[_soilt]	Gridbox methane flux from wetland fraction using soil respiration as substrate (kg m^{-2} s^{-1}).	
substr_ch4	Carbon in substrate pool used by methanogens for each soil methane layer (kg m^{-2}). Only available if *l_ch4_microbe* = TRUE.	ch4layer
mic_ch4	Carbon in methanogenic biomass for each soil methane layer (kg m^{-2}). Only available if *l_ch4_microbe* = TRUE.	ch4layer
mic_act_ch4	Activity level of methanogenic biomass for each soil methane layer (-). Only available if *l_ch4_microbe* = TRUE.	ch4layer
acclim_ch4	Acclimation factor for methanogenic processes in each soil methane layer (-). Only available if *l_ch4_microbe* = TRUE.	ch4layer

9.11 Soil nitrogen and related fluxes

Name	Description	Dimensions
`n_soil_gb`	Gridbox total soil nitrogen (organic and inorganic) (kg m^{-2}). Only available if *soil_bgc_model* = 2 or 3.	
`ns`	Gridbox organic nitrogen in each soil pool and each soil biogeochemistry layer (kg m^{-2}).	cspool,cslayer
`ns_gb`	Gridbox soil organic nitrogen (kg m^{-2}). Only available if *soil_bgc_model* = 2 or 3.	
`n_bio_gb`	Gridbox soil nitrogen in biomass pool (kg m^{-2}). Only available if *soil_bgc_model* = 2 or 3.	
`n_dpm_gb`	Gridbox soil nitrogen in decomposable plant material pool (kg m^{-2}). Only available if *soil_bgc_model* = 2 or 3.	
`n_rpm_gb`	Gridbox soil nitrogen in resistant plant material pool (kg m^{-2}). Only available if *soil_bgc_model* = 2 or 3.	
`n_hum_gb`	Gridbox soil nitrogen in humus pool (kg m^{-2}). Only available if *soil_bgc_model* = 2 or 3.	
`n_amm_gb`	Gridbox soil nitrogen in ammonium pool (kg m^{-2}). Only available if *soil_bgc_model* = 3.	
`n_nit_gb`	Gridbox soil ammonium in ammonium pool (kg m^{-2}). Only available if *soil_bgc_model* = 3.	
`n_inorg_gb`	Gridbox soil inorganic nitrogen (kg m^{-2}). Only available if *soil_bgc_model* = 2 or 3.	
`n_inorg_avail_pft`	PFT inorganic nitrogen pool that is available for plant uptake for each soil biogeochemistry layer (kg m^{-2}).	npft
Soil nitrogen fluxes		
`immob_n`	Soil nitrogen immobilisation in each soil pool and each soil biogeochemistry layer (kg m^{-2} (360days)$^{-1}$).	cspool,cslayer
`immob_n_gb`	Gridbox mean soil nitrogen immobilisation (kg m^{-2} (360days)$^{-1}$).	
`immob_n_pot`	Soil potential nitrogen immobilisation in each soil pool and each soil biogeochemistry layer (kg m^{-2} (360days)$^{-1}$).	cspool,cslayer
`immob_n_pot_gb`	Gridbox mean potential soil nitrogen immobilisation (kg m^{-2} (360days)$^{-1}$).	
`minl_n_pot`	Soil potential nitrogen mineralisation in each pool soil pool and each soil biogeochemistry layer (kg m^{-2} (360days)$^{-1}$).	cspool,cslayer
`minl_n_pot_gb`	Gridbox mean potential soil nitrogen mineralisation (kg m^{-2} (360days)$^{-1}$).	
`minl_n`	Soil nitrogen mineralisation in each pool soil pool and each soil biogeochemistry layer (kg m^{-2} (360days)$^{-1}$).	cspool,cslayer
`minl_gb`	Gridbox mean soil nitrogen mineralisation (kg m^{-2} (360days)$^{-1}$).	
`n_miner_gb`	Gribox rate of net mineralisation of soil N (kg m^{-2} s^{-1}). Only available with the ECOSSE soil model (*soil_bgc_model* = 3).	
`n_nitrif_gb`	Gridbox mean rate of nitrification, expressed as N (kg m^{-2} s^{-1}).	
`n_denitrif_gb`	Gridbox mean rate of denitrification, expressed as N (kg m^{-2} s^{-1}).	
`n2o_nitrif_gb`	Gridbox mean N in N_2O lost during nitrification, including partial nitrification (kg m^{-2} s^{-1}).	
`n2o_part_nitrif_gb`	Gridbox mean N in N_2O lost by partial nitrification (kg m^{-2} s^{-1}).	
`n2_denitrif_gb`	Gridbox mean N in N_2 lost from soil during denitrification (kg m^{-2} s^{-1}).	
`n2o_denitrif_gb`	Gridbox mean N in N_2O lost during denitrification (kg m^{-2} s^{-1}).	
`n2o_soil_gb`	Gridbox mean N in N_2O flux from soil to atmosphere (kg m^{-2} s^{-1}).	
`no_soil_gb`	Gridbox mean N in NO flux from soil to atmosphere (kg m^{-2} s^{-1}).	
`n_fertiliser_gb`	Gridbox nitrogen addition from fertiliser (kg N m^{-2} (360days)$^{-1}$).	
`n_gas_gb`	Gridbox mean mineralised nitrogen gas emissions (kg m^{-2} (360days)$^{-1}$).	

continues on next page

Table 6 – continued from previous page

Name	Description	Dimensions
n_leach	Gridbox leached nitrogen loss term (kg N m^{-2} s^{-1}).	
n_loss	Gridbox nitrogen loss term (fixed fraction of n_inorg) (kg N m^{-2} (360days)$^{-1}$).	

9.12 Fire

Name	Description	Dimensions
Fire indices		
fire_mcarthur	McArthur Forest Fire Danger Index (No units)	
fire_canadian	Canadian Fire Weather Index (No units).	
fire_canadian_ffmc	Canadian Fire Weather Index- Fine Fuel Moisture Code (No units).	
fire_canadian_dmc	Canadian Fire Weather Index- Duff Moisture Code (No units).	
fire_canadian_dc	Canadian Fire Weather Index- Drought Code (No units).	
fire_canadian_isi	Canadian Fire Weather Index- Initial Spread Index (No units).	
fire_canadian_bui	Canadian Fire Weather Index- Build-up Index (No units).	
fire_nesterov	Nesterov Fire Index (No units).	
Burnt area		
burnt_area	PFT burnt area fraction (s^{-1}).	npft
burnt_area_gb	Gridbox mean burnt area fraction (s^{-1}).	
Fire emissions		
emitted_carbon	PFT emitted carbon flux (kg m^{-2} s^{-1}).	npft
emitted_carbon_gb	Gridbox mean emitted carbon flux (kg m^{-2} s^{-1}).	
emitted_carbon_DPM	Decomposable Plant Material emitted carbon flux (kg m^{-2} s^{-1}).	
emitted_carbon_RPM	Resistant Plant Material emitted carbon flux (kg m^{-2} s^{-1}).	
fire_em_CO2_gb	Gridbox mean fire CO2 emission flux (kg m^{-2} s^{-1}).	
fire_em_CO2_DPM	Decomposable Plant Material fire CO2 emission flux (kg m^{-2} s^{-1}).	
fire_em_CO2_RPM	Resistant Plant Material fire CO2 emission flux (kg m^{-2} s^{-1}).	
fire_em_CO_gb	Gridbox mean fire CO emission flux (kg m^{-2} s^{-1}).	
fire_em_CO_DPM	Decomposable Plant Material fire CO emission flux (kg m^{-2} s^{-1}).	
fire_em_CO_RPM	Resistant Plant Material fire CO emission flux (kg m^{-2} s^{-1}).	
fire_em_CH4_gb	Gridbox mean fire CH4 emission flux (kg m^{-2} s^{-1}).	
fire_em_CH4_DPM	Decomposable Plant Material fire CH4 emission flux (kg m^{-2} s^{-1}).	
fire_em_CH4_RPM	Resistant Plant Material fire CH4 emission flux (kg m^{-2} s^{-1}).	
fire_em_NOx_gb	Gridbox mean fire NOx emission flux (kg m^{-2} s^{-1}).	
fire_em_NOx_DPM	Decomposable Plant Material fire NOx emission flux (kg m^{-2} s^{-1}).	
fire_em_NOx_RPM	Resistant Plant Material fire NOx emission flux (kg m^{-2} s^{-1}).	
fire_em_SO2_gb	Gridbox mean fire SO2 emission flux (kg m^{-2} s^{-1}).	
fire_em_SO2_DPM	Decomposable Plant Material fire SO2 emission flux (kg m^{-2} s^{-1}).	
fire_em_SO2_RPM	Resistant Plant Material fire SO2 emission flux (kg m^{-2} s^{-1}).	
fire_em_OC_gb	Gridbox mean fire OC emission flux (kg m^{-2} s^{-1}).	
fire_em_OC_DPM	Decomposable Plant Material fire OC emission flux (kg m^{-2} s^{-1}).	
fire_em_OC_RPM	Resistant Plant Material fire OC emission flux (kg m^{-2} s^{-1}).	
fire_em_BC_gb	Gridbox mean fire BC emission flux (kg m^{-2} s^{-1}).	
fire_em_BC_DPM	Decomposable Plant Material fire BC emission flux (kg m^{-2} s^{-1}).	
fire_em_BC_RPM	Resistant Plant Material fire BC emission flux (kg m^{-2} s^{-1}).	
fire_em_CO2	PFT fire CO2 emission flux (kg m^{-2} s^{-1}).	npft
fire_em_CO	PFT fire CO emission flux (kg m^{-2} s^{-1}).	npft
fire_em_CH4	PFT fire CH4 emission flux (kg m^{-2} s^{-1}).	npft
fire_em_NOx	PFT fire NOx emission flux (kg m^{-2} s^{-1}).	npft
fire_em_SO2	PFT fire SO2 emission flux (kg m^{-2} s^{-1}).	npft
fire_em_OC	PFT fire OC emission flux (kg m^{-2} s^{-1}).	npft

continues on next page

Table 7 – continued from previous page

Name	Description	Dimensions
`fire_em_BC`	PFT fire BC emission flux (kg m^{-2} s^{-1}).	npft

9.13 Crops

These variables are only available when *ncpft* > 0.

Name	Description	Dimensions
`croplai`	Crop PFT leaf area index (LAI).	ncpft
`cropcanht`	Crop PFT canopy height (m).	ncpft
`cropleafc`	Crop PFT carbon in leaf parts (kg m^{-2}).	ncpft
`cropstemc`	Crop PFT carbon in stem parts (kg m^{-2}).	ncpft
`croprootc`	Crop PFT root carbon pool (kg m^{-2}).	ncpft
`cropreservec`	Crop PFT carbon in stem reserve pool (kg m^{-2}).	ncpft
`cropharvc`	Crop PFT carbon in harvested parts (kg m^{-2}).	ncpft
`cropyield`	Crop PFT yield carbon (kg m^{-2}). `cropyield` is zero in every timestep where there is no harvest, so this variable will mostly be used with *output_type* set to ‘A’ or ‘X’.	ncpft
`cropdvi`	Crop PFT developmental index (DVI).	ncpft
`cropsowdate`	PFT sowing date (1.0 to 365.0). If *l_prescsow* = FALSE then this will always be 0.	ncpft
`harvest_counter`	1 in a timestep where crop is harvested, 0 otherwise. When used with *output_type* = ‘A’, will count the number of harvests since the beginning of the run.	ncpft
`harvest_trigger`	Indicates which condition triggered the harvest: 0. No harvest in timestep. 1. Crop reached maturity (DVI=2). 2. Crop leaf area index became too high (LAI>15). 3. Crop has flowered (DVI>1) and temperature in the second soil layer dropped below *t_mort_io*. 4. Crop has flowered (DVI>1), `cropharvc` > 0 and (`croprootc` + `cropleafc` + `cropstemc` + `cropreservec`) < *initial_carbon_io*. 5. *l_prescsow* = T, crop has emerged (DVI>=0), and the day of year is the day before the sowing date.	ncpft

9.14 Trace gas concentrations and fluxes

Name	Description	Dimensions
`co2_mmr`	Concentration of atmospheric CO2, expressed as a mass mixing ratio.	
`flux_o3_stom`	PFT flux of O3 to stomata (nmol m^{-2} s^{-1}).	npft
`o3_exp_fac`	PFT ozone exposure factor.	npft
`acetone`	PFT monoterpene emission flux (kg m^{-2} s^{-1}).	npft
`acetone_gb`	Gridbox mean monoterpene emission flux (kg m^{-2} s^{-1}).	
`isoprene`	PFT isoprene emission flux (kg m^{-2} s^{-1}).	npft
`isoprene_gb`	Gridbox mean isoprene emission flux (kg m^{-2} s^{-1}).	
`methanol`	PFT methanol emission flux (kg m^{-2} s^{-1}).	npft
`methanol_gb`	Gridbox mean methanol emission flux (kg m^{-2} s^{-1}).	
`terpene`	PFT monoterpene emission flux (kg m^{-2} s^{-1}).	npft
`terpene_gb`	Gridbox mean monoterpene emission flux (kg m^{-2} s^{-1}).	

9.15 Water resources

Name	Description	Dimensions
`water_demand`	Demand for water across all water resource sectors (kg). Only available if *l_water_resources* = TRUE.	
`demand_domestic`	Demand for water for domestic use (kg). Only available if *l_water_domestic* = TRUE.	
`demand_environment`	Demand for water for environmental use (kg). Only available if *l_water_environment* = TRUE.	
`demand_industry`	Demand for water for industrial use (kg). Only available if *l_water_industry* = TRUE.	
`demand_irrigation`	Demand for water for irrigation (kg). Only available if *l_water_irrigation* = TRUE.	
`demand_livestock`	Demand for water for livestock use (kg). Only available if *l_water_livestock* = TRUE.	
`demand_transfers`	Demand for water for transfers (kg). Only available if *l_water_transfers* = TRUE.	
`demand_rate_domestic`	Demand rate for water for domestic use (kg s^{-1}). Only available if *l_water_irrigation* = TRUE.	
`demand_rate_industry`	Demand rate for water for industrial use (kg s^{-1}). Only available if *l_water_industry* = TRUE.	
`demand_rate_livestock`	Demand rate for water for livestock use (kg s^{-1}). Only available if *l_water_livestock* = TRUE.	
`demand_rate_transfers`	Demand rate for water for transfers (kg s^{-1}). Only available if *l_water_transfers* = TRUE.	
`irrig_water`	Water applied as irrigation (kg m^{-2} s^{-1}). Only available if *l_irrig_dmd* = TRUE.	

9.16 Urban

Name	Description	Dimensions
`wrr`	Urban morphology: Repeating width ratio (or canyon fraction, W/R). Calculated if *l_urban_empirical* = TRUE, otherwise it will be the same as the input value (see *List of urban properties*).	
`hwr`	Urban morphology: Height-to-width ratio (H/W). See for `wrr` above. Only used by MORUSES.	
`hgt`	Urban morphology: Building height (H). See for `wrr` above. Only used by MORUSES.	

9.17 IMOGEN

Name	Description	Dimensions
`c_emiss_out`	Prescribed carbon emissions in IMOGEN (Gt / year). This is a global value repeated over all land points and is only available when IMOGEN is switched on.	
`d_land_atmos_co2`	Change in atmospheric CO2 concentration from land-atmosphere. CO2 feedbacks in IMOGEN (ppm / year). This is a global value repeated over all land points and is only available when IMOGEN is switched on.	
`d_ocean_atmos`	Change in atmospheric CO2 concentration from ocean-atmosphere CO2 feedbacks in IMOGEN (ppm / year). This is a global value repeated over all land points and is only available when IMOGEN is switched on.	

9.18 Grid and indexing variables

Name	Description	Dimensions
`grid_area`	Gridbox surface area (m^2). Only available if *`l_water_irrigation`* = TRUE.	land+sea
`grid_area_rp`	River routing gridbox surface area (m^2). Only available if *`l_rivers`* = TRUE. Note that this is defined on the river routing model grid, not on the land point grid.	np_rivers
`land_index`	Index (gridbox number) of land points.	
`lice_index`	Index (gridbox number) of land ice points.	
`soil_index`	Index (gridbox number) of soil points.	
`tile_index`	Index (gridbox number) of land points with each surface type.	ntype

FORTRAN NAMELIST INDEX

INDEX