User manual for ECHAM6

Sebastian Rast[1], Renate Brokopf, Monika Esch,
Veronika Gayler, Ingo Kirchner, Luis Kornblüh,
Andreas Rhodin, Uwe Schulzweida

February 22, 2012, (2012.02-22), version echam–6.1.00–guide–2.0

[1]Max Planck Institute of Meteorology, Hamburg, e–mail: sebastian.rast@zmaw.de

Contents

List of Tables

Chapter 1

Introduction

The ECHAM6 model is a program for the interactive calculation of the general circulation. This manual contains a user guide of ECHAM6 (chapter 2) including a description of the compilation procedure on the supercomputer platform blizzard at DKRZ Hamburg (section 2.1), a description of the input namelists (section 2.2), input files (section 2.3), and ouput files (section 2.4), a description of example run scripts (section 2.5), and postprocessing scripts (section 2.6). We restrict our description to the supercomputer platform blizzard at DKRZ in Hamburg. Performing a simulation on other computer platforms requires the same input data, but the compiling procedure and the directory structure for output in particular, will be different.
Chapter 3 contains a short description of the code of ECHAM6 and is intended to be a guide for people who work with the source code of the atmosphere part of ECHAM6. An introduction to the ECHAM6–code with explanations will become available in form of a lecture soon ("Using and programming ECHAM6 — a first introduction").
This description is valid for version echam–6.1.00.

Chapter 2

User guide

2.1 Compiling ECHAM6

The following commands have to be executed in order to compile the ECHAM6 model on the supercomputer platform blizzard at Deutsches Klimarechenzentrum (DKRZ):

- Checkout a model version with command:
  `svn checkout http://svn.zmaw.de/svn/echam6/tags/echam-<tag_number>`
  of a certain tagged version.
- Load the appropriate compiler version used for ECHAM6, e.g.:
  `module load IBM/xlf13.1.0.2`
- Go into the directory `echam-<tag_number>` and execute the command
  `./configure --with-openmp`
- Start the actual compilation with the command
  `make`

2.2 Input namelists

2.2.1 Input namelists in file `namelist.echam`

In Fortran, you can provide the values of input variables that are organized in namelists, specifying name and value of each variable. Several namelists are used to specify the input of ECHAM6. Some of the namelists are for the atmospheric part and have to be written into the file `namelist.echam`, others determine input variables of the land surface model JSBACH and have to be written into `namelist.jsbach`. The atmospheric part can accept the following namelists in `namelist.echam` (alphabetical order):

`cfdiagctl`: CFMIP dignostics.

`co2ctl`: interactive CO_2 budget calculation.

`columnctl`: single column model.

`debugsctl:` creates a stream for grid point variables that can be written to output easily (for debugging).

`dyctl:` parameters for atmosphere dynamics.

`gwsctl:` gravity wave parameterisation.

`hratesctl:` diagnostic of heating rates.

`mvstreamctl:` variables controlling output of mean values.

`nmictl:` normal mode analysis of waves.

`ndgctl:` variables which are related to the nudging of the model, i.e. to the relaxation method constraining the meteorological variables divergence, vorticity, temperature and pressure to externally given values.

`parctl:` parameters concerning the parallel configuration of model.

`physctl:` variables related to the physics calculation like switching on/off radiation, diffusion, convection, surface exchange, ...

`radctl:` variables for controlling the radiation calculation.

`runctl:` contains variables concerning the start and the end of a simulation.

`submdiagctl:` submodel diagnostics.

`submodelctl:` namelists for registration of submodels in ECHAM6.

`tdiagctl:` tendency diagnostic.

The syntax for each namelist in `namelist.echam` is:

Listing 2.1: namelist syntax

```
& <namelist name>
  <varname> = <value>
/
```

Remark: The mere presence of a certain variable in a certain namelist does not mean that the action associated with this variable really works properly or works at all.
Variables describing repeated events have a special format (type "special" in the following tables):
{interval}, {unit}, {adjustment}, {offset}
where {interval} is a positive integer number, {unit} is one of 'steps', 'seconds', 'minutes', 'hours', 'days', 'months', 'years', {adjustment} is one of 'first', 'last', 'exact', 'off', and {offset} is an integer number giving the offset with respect to the initial date of the simulation in seconds. A detailed description of the control of time events can be found in the lecture "Using and programming ECHAM6 — a first introduction" by S. Rast. The variable list is given in alphabetical order even if the most important variables are not at the first place in this case.

2.2.1.1 Namelist `cfdiagctl`

This namelists contains only one parameter to switch on or off the CFMIP diagnostics of 3–dimensional fluxes.

Table 2.1: Namelist `cfdiagctl`

Variable	type	Explanation	default
`locfdiag`	logical	switches on/off CFMIP diagnostic output of convective mass flux and 3-D radiation fluxes	`.FALSE.`

2.2.1.2 Namelist `co2ctl`

This namelist controls the behaviour of the CO_2 submodel. This submodel is not a simple submodel like the transport of some gas phase species would be because the CO_2 module interacts with the JSBACH surface and vegetation model. In this namelist, the behaviour of the CO_2 submodel in the atmosphere simulated by ECHAM6 and the interaction with the ocean and soil simulated by JSBACH can be controlled.

Table 2.2: Namelist `co2ctl`

Variable	type	Explanation	default
`lco2_flxcor`	logical	switches on/off flux correction for exact mass balance	`.TRUE.`
`lco2_mixpbl`	logical	switches on/off CO_2 mixing in planetary boundary layer	`.TRUE.`
`lco2_2perc`	logical	switches on/off limitation of relative CO_2 tendency to 2%	`.FALSE.`
`lco2_emis`	logical	switches on/off reading prescribed CO_2 emissions from a file	`.FALSE.`
`lco2_clim`	logical	switches on/off treating the CO_2 concentration as a climatological quantity not being transported	`.FALSE.`
`lco2_scenario`	logical	switches on/off reading CO_2 concentrations from a certain greenhouse gas scenario	`.FALSE.` but `.TRUE.` if `ighg=1` and `lco2=.FALSE.`

2.2.1.3 Namelist `columnctl`

This namelist controls the behaviour of the column model. It is an old version of the column model that is not working and has to be replaced by a newer version.

2.2.1.4 Namelist `debugsctl`

The debug stream is meant to provide a quick and easy tool to the user of ECHAM6 that allows him to write any 2d– or 3d–gridpoint variable to an extra stream for debugging. A detailed

description can be found in the document cr2009_03_31 (can be provided by S. Rast (sebastian.rast@zmaw.de)).

Table 2.3: Namelist `debugsctl`

Variable	type	Explanation	default
`nddf`	integer	number of 3d–fields created in addition to the default fields	`0`
`nzdf`	integer	number of 2d–fields created in addition to the default fields	`0`
`putdebug_stream`	special	output frequency of debug stream	`6, 'hours', 'first', 0`

2.2.1.5 Namelist `dynctl`

With the help of these namelist parameters, the (large scale) dynamics of the atmosphere can be controlled.

Table 2.4: Namelist `dynctl`

Variable	type	Explanation	default
`apsurf`	double prec	fixed global mean of surface pressure in Pa fixing the mass of the dry atmosphere	`98550.0`
`damhih`	double prec	extra diffusion in the middle atmosphere	`1000.`

table continued on next page

Table 2.4: `dynctl` — continued

`dampth`	double prec	damping time in hours for the horizontal diffusion of vorticity (linear square Laplacian), divergence, and temperature. Depends on the spectral resolution `nn`	`nn=21, lmidatm=.FALSE.: 6.0, 15.0 if nlev=11`
			`nn=21, lmidatm=.TRUE.: 192.0`
			`nn=31: 12.0, 15.0 if nlev=11`
			`nn=42: 9.0`
			`nn=63: 7.0`
			`nn=85: 5.0`
			`nn=106: 3.0`
			`nn=127: 1.5`
			`nn=159: 2.0`
			`nn=213: 2.0`
			`nn=255: 0.5`
			`nn=319: 1.0`

`diagdyn`	special	frequency for diagnostic output of quantities describing the dynamics of the atmosphere	`5, 'days', 'off', 0`
`diagvert`	special	frequency for special (all layers) diagnostic output of quantities describing the dynamics of the atmosphere	`5, 'days', 'off', 0`
`enspodi`	double prec	factor by which upper sponge layer coefficient is increased from one layer to the adjacent layer above	`1.0`
`enstdif`	double prec	factor by which stratospheric horizontal diffusion is increased from one layer to the adjacent layer above	`1.0`
`eps`	double prec	coefficient in the Robert–Asselin time filter	`0.1`
`hdamp`	double prec	damping factor for strong stratospheric damping	`1.0`
`ldiahdf`	logical	switches on/off statistical analysis of horizontal diffusion	`.FALSE.`
`lumax`	logical	switches on/off the printing of information on maximum wind speeds	`.FALSE.`
`lzondia`	logical	purpose unknown	`.FALSE.`

table continued on next page

Table 2.4: `dynctl` — continued

`nlvspd1`	integer	model layer index of uppermost layer of upper sponge	`1`
`nlvspd2`	integer	model layer index of lowest layer of upper sponge	`1`
`nlvstd1`	integer	model layer index of uppermost layer at which stratospheric horizontal diffusion is enhanced	`1`
`nlvstd2`	integer	model layer index of lowest layer at which stratospheric horizontal diffusion is enhanced	`1`
`ntrn(1:nlev)`	integer	layer and resolution dependent critical wave numbers for strong stratospheric damping	see `setdyn.f90`
`spdrag`	double prec	coefficient for upper sponge layer in 1/s	`0.0`, if `lmidatm=.TRUE.`: 0.926×10^{-4} (see Tab. 2.12
`vcheck`	double prec	threshold value for check of high windspeed in m/s	`200.0` if `lmidatm=.TRUE.` (see Tab. 2.12: `235.0`
`vcrit`	double prec	critical velocity above which horizontal diffusion is enhanced in m/s. Depends on the spectral resolution `nn`	`nn=106`: 68.0 all other `nn`: 85.0

2.2.1.6 Namelist `gwsctl`

This namelist controls the settings for the gravity wave drag parameterization.

Table 2.5: Namelist `gwsctl`

Variable	type	Explanation	default
`emiss_lev`	integer	model layer index counted from the surface at which gravity waves are emitted. This number depends on the vertical resolution and corresponds to a model layer that is at roughly 600 hPa in the standard atmosphere.	`nlev=39`: 7 `nlev=199`: 26 all other `nlev`: 10

table continued on next page

Table 2.5: `gwsctl` — continued

`front_thres`	double prec	minimum value of the frontogenesis function for which gravity waves are emitted from fronts in $(K/m)^2/h$	`0.12`
`iheatcal`	integer	controls upper atmosphere processes associated with gravity waves: `iheatcal=1`: calculate heating rates and diffusion coefficient in addition to momentum flux deposition `iheatcal=2`: momentum flux deposition only	`1`
`kstar`	double prec	typical gravity wave horizontal wave number	5×10^{-5}
`lat_rmscon_hi`	double prec	latitude above which extratropical gravity wave source is used. Is only relevant if `lrmscon_lat=.TRUE.`	`10.0`
`lat_rmscon_lo`	double prec	latitude below which tropical gravity wave source is used. Is only relevant if `lrmscon_lat=.TRUE.`. There is a linear interpolation between `lat_rmscon_lo` and `lat_rmscon_hi` degrees N and S, respectively, between the values given by `rmscon_lo` (associated with the tropical gravity wave parameterization) and `rmscon_hi` associated with the extratropical gravity wave parameterization	`5.0`
`lextro`	logical	switches on/off the Doppler spreading extrowave parameterization by Hines	`.TRUE.`
`lfront`	logical	switches on/off gravity waves emerging from fronts and the background. Parameterization by Charron and Manzini	`.TRUE.`

table continued on next page

Table 2.5: `gwsctl` — continued

`lozpr`	logical	switches on/off the background enhancement of gravity waves associated with precipitation by Manzini et al.. Does not work with ECHAM6.	`.FALSE.`
`lrmscon_lat`	logical	switches on/off latitude dependent `rmscon` as defined in `setgws`. May be overwritten by `lfront=.TRUE.` or `lozpr=.TRUE.`	`.FALSE.`
`m_min`	double prec	minimum bound in vertical wave number	`0.0`
`pcons`	double prec	factor for background enhancement associated with precipitation	`4.75`
`pcrit`	double prec	critical precipitation value above which root mean square gravity wave wind enhancement is applied in mm/d	`5.0`
`rms_front`	double prec	root mean square frontal gravity wave horizontal wave number in 1/m	`2.0`
`rmscon`	double prec	root mean square gravity wave wind at lowest layer in m/s	`1.0`
`rmscon_hi`	double prec	root mean square gravity wave wind at lowest layer in m/s for extratropical gravity wave source. Is only relevant if `lrmscon_lat=.TRUE.`	`1.0`

`rmscon_lo`	double prec	root mean square gravity wave wind at lowest layer in m/s for tropical gravity wave source. Is only relevant if `lrmscon_lat=.TRUE.`. Depends on the spectral resolution `nn`	`nn=31`: `1.0`
			`nn=63`: `1.2`
			`nn=127`: `1.05` but `1.1` if `lcouple=.TRUE.` (see Tab. 2.12)
			any other `nn`: 1.1

2.2.1.7 Namelist `hratesctl`

This namelist is obsolete since its functionality is included in `tdiagctl` (see section 2.2.1.17).

2.2.1.8 Namelist `mvstreamctl`

Using this namelist, the online calculation of mean values of non–accumulated grid point variables of any ouput stream is possible. For each stream, you can ask for one additional stream containing the mean values of a subset of variables of this stream. The namelist `mvstreamctl` controls which output streams will be doubled. The ouput of mean values of trace species concentrations are written to the ouput stream `tracerm`. This namelist works together with the namelist `mvctl` described in section 2.2.10.1. Additional documentation can be found in cr2010_07_28 provided by S. Rast (sebastian.rast@zmaw.de).

Table 2.6: Namelist `mvstreamctl`

Variable	type	Explanation	default
`m_stream_name(256,1:50)`	character	List of names of streams for the elements of which mean values shall be calculated. Note that a maximum of 50 output stream is allowed (including the mean value streams).	empty string

2.2.1.9 Namelist `ndgctl`

This namelist controls all variables that are relevant for nudging, i.e. relevant for a simulation mode in which the spectral 3d–temperature, vorticity, divergence, surface pressure, and surface temperature can be constrained to external fields obtained e.g. from the assimilation of observations. It has to be underlined that constraining the surface temperature may lead to wrong sea ice coverage since the presence of sea ice is diagnosed from the surface temperature directly without taking into account any hysteresis effects (see cr2008_08_11 by S. Rast, sebastian.rast@zmaw.de).

Table 2.7: Namelist `ndgctl`

Variable	type	Explanation	default
`dt_nudg_start(1:6)`	integer	defines the beginning of the nudging in the experiment. Is of the form yy,mo,dy,hr,mi,se (year, month, day, hour, minute, second)	`0,0,0,0,0,0`
`dt_nudg_stop(1:6)`	integer	defines the date at which nudging stops in a simulation. Is of the form yy,mo,dy,hr,mi,se (year, month, day, hour, minute, second)	`0,0,0,0,0,0`

table continued on next page

Table 2.7: `ndgctl` — continued

`inudgformat`	integer	format of nudging input files `inudgformat` = 0: old CRAY format input files `inudgformat` = 2: netcdf format input file	0
`ldamplin`	logical	linear damping (`ldamplin` = `.true.`) or damping with a parabolic function (`ldamplin` = `.false.`) of the nudging efficiency between two synoptic times at which nudging data sets are given	`.true.`
`lnudgdbx`	logical	`.true.` for additional diagnostic output about nudging, `.false.` otherwise	`.false.`
`lnudgcli`	logical	`lnudgcli` = `.true.`: ECHAM6 ignores the information about the year in the nudging data file and reads nudging data in a cyclic way. Consequently, for each model year, the same nudging data are read. `lnudgcli` = `.false.`: The information about the year is included in the nudging procedure, the data to which the model is constrained depend on the year.	`.false.`
`lnudgfrd`	logical	`lnudgfrd` = `.true.`: normal mode filtering is done at reading the data `lnudgfrd` = `.false.`: normal mode filtering is done elsewhere. Works only together with `lnmi=.true.`	`.false.`
`lnudgimp`	logical	`lnudgimp` = `.true.`: implicit nudging `lnudgimp` = `.false.`: explicit nudging	`.true.`

table continued on next page

Table 2.7: `ndgctl` — continued

`lnudgini`	logical	`lnudgini` = .`false`.: ECHAM6 starts or restarts a simulation for a certain experiment from the date given in the namelist by `dt_start` or the restart date in the restart file `lnudgini` = .`true`.: If `lresume` = .`false`., the model starts the simulation at the date of the first nudging data set being in the nudging files the names of which correspond to `dt_nudge_start`. There must be nudging files having a file name corresponding to `dt_nudge_start`. If `lresume=.true.`, the model starts its run at the first date being in the nudging data files the file names of which correspond to the `next_date` (next time step) of the rerun date.	.`false`.
`lnudgpat`	logical	`lndgpat` = .`true`.: pattern nudging. Does not work properly, to be removed `lndgpat` = .`false`: otherwise	.`false`.
`lnudgwobs`	logical	.`true`. for storing additional nudging reference fields, .`false`. otherwise	.`false`.
`lsite`	logical	switches on/off the Systematic Initial Tendency Error diagnostic	.`false`.
`ltintlin`	logical	`ltintlin` = .`true`.: linear time interpolation `ltintlin` = .`false`. for cubic spline time interpolation between two synoptic times at which nudging data sets are given	.`true`.
`ndg_file_div(256)`	character	file name template for the file containing the nudging data for the divergence	—
`ndg_file_nc(256)`	character	file name template for netcdf format file containing all nudging data (temperature, logarithm of surface pressure, divergence and vorticity)	—
`ndg_file_sst(256)`	character	file name template for file containing the sea surface temperature	—
`ndg_file_stp(256)`	character	file name template for the file containing the nudging data for the temperature and the logarithm of the surface pressure	—
`ndg_file_vor(256)`	character	file name template for the file containing the nudging data for the vorticity	—

table continued on next page

Table 2.7: `ndgctl` — continued

`ndg_freez`	double prec	temperature at which sea water is assumed to freeze in Kelvin	`271.65`
`nsstinc`	integer	treatment of the sea surface temperature (sst): read new sst data set each `nsstinc` hours. A value of 0 means that sst is not used and prevents the model to produce too low sea ice coverage when nudging since sea ice would be detected only if temperatures drop below `ndg_freez`	`0`
`nsstoff`	integer	read the first sst data at hour `nsstoff` after the beginning of the nudging	`12`
`nudgd(1:80)`	double prec	the relaxation time for each model layer for the nudging of the spectral divergence is given by $1/(\texttt{nudgd} \times 10^{-5})s$. Note the maximum of 80 layers!	`0.5787(1:80)`/s corresponding to 48 hours
`nudgdamp`	double prec	the nudging between two synoptic times will be reduced to `nudgdamp`. Consequently, `nudgdamp`=1.0 means that nudging will be effective at 100% at every time step, `nudgdamp`=0.0 means that the nudging will be switched off somewhere between two synoptic times at which nudging data are available	`1.0`
`nudglmax`	integer	highest index of the model layer at which nudging is performed. Note the maximum of 80 layers!	`80`
`nudglmin`	integer	lowest index of the model layer at which nudging is performed	`1`
`nudgp`	double prec	the relaxation time for the nudging of the logarithm of the surface pressure is given by $1/(\texttt{nudgp} \times 10^{-5})$ s	`1.1574`/s correspondig to 24 hours
`nudgdsize`	double prec	fraction of the synoptic time interval after which only the fraction `nudgdamp` is applied in the nudging procedure. If `nudgdsize` < 0.5, the minimum is reached after a fraction of `nudgdsize` of the synoptic time interval. This minimum nudging level is then maintained until the model time reaches the next synoptic time step minus the fraction `nudgdsize` of the synoptic time interval. Then, the nudging strength is starting to increase again	`0.5`

table continued on next page

Table 2.7: `ndgctl` — continued

`nudgsmax`	integer	highest nudged wavenumber. Note the restriction to model resolution not higher than T106!	`106`
`nudgsmin`	integer	Index of lowest nudged wavenumber minus one. This means, that with `nudgsmin` = 0, the spectral coefficient 0 (global average) is not nudged	`0`
`nudgt(1:80)`	double prec	the relaxation time for each model layer for the nudging of the spectral temperature is given by $1/(\texttt{nudgt} \times 10^{-5})s$. Note the maximum of 80 layers!	`1.1574(1:80)`/s corresponding to 24 hours
`nudgtrun`	integer	mode of selection of spectral coefficients for nudging (see `mo_nudging_init.f90`)	`0`
`nudgv(1:80)`	double prec	the relaxation time for each model layer for the nudging of the spectral vorticity is given by $1/(\texttt{nudgv} \times 10^{-5})s$. Note the maximum of 80 layers!	`4.6296(1:80)`/s corresponding to 6 hours

2.2.1.10 Namelist `nmictl`

This is the namelist to control the normal mode analysis.

Table 2.8: Namelist `nmictl`

Variable	type	Explanation	default
`dt_nmi_start(1:6)`	integer	start date of the NMI procedure. Is of the form yy,mo,dy,hr,mi,se (year, month, day, hour, minute, second)	`0,0,0,0,0,0`
`lnmi_cloud`	logical	run initialization including clouds	`.TRUE.`
`ntdia`	integer	number of time steps of accumulation interval for diabatic tendencies	`8`
`ntiter`	integer	number of time steps in an iteration interval	`2`
`ntpre`	integer	number of time steps of pre–integration interval	`2`
`pcut`	double prec	cut off period in hours (used for nudging)	`12.0`
`pcutd`	double prec	cut off period in hours (used for initialization)	`6.0`

2.2.1.11 Namelist `parctl`

This namelist controls the parallelization of the ECHAM6 program.

Table 2.9: Namelist `parctl`

Variable	type	Explanation	default
`db_host(132)`	character	hostname of db server of timing database	`''`
`lunitrans`	logical	switches on/off the use of the UNITRANS module for transposes	`.FALSE.`
`lunitrans_datatypes`	logical	switches on/off MPI data types in UNITRANS transposes	`.TRUE.`
`lunitrans_debug`	logical	switches on/off the debugging of the UNITRANS calls	`.FALSE.`
`network_logger(132)`	character	hostname for network logging	`''`
`nproca`	integer	number of processors for set A division of earth	`1`
`nprocb`	integer	number of processors for set B division of earth	`1`
`nprocio`	integer	number of processors used for I/O, not yet functional	`0`

2.2.1.12 Namelist `physctl`

This namelist controls the physics calculations in ECHAM6. These are mainly calculations in the grid point space with parametrized equations for convection, diffusion, gravity waves, and the exchange of energy and mass at the surface of the earth.

Table 2.10: Namelist `physctl`

Variable	type	Explanation	default
`iconv`	integer	switch for convection scheme: `iconv` = 1: Nordeng `iconv` = 2: Tiedtke `iconv` = 3: Hybrid	`1`
`lcdnc_progn`	logical	switches on/off prognostic cloud droplet number concentration	`.false.`
`lcond`	logical	switches on/off large scale condensation scheme	`.true.`
`lconv`	logical	switches on/off convection	`.true.`
`lconvmassfix`	logical	switches on/off aerosol mass fixer in convection (obsolete?)	`.true.`
`lcover`	logical	switches on/off Tompkins cloud cover scheme	`.true.`
`lgwdrag`	logical	switches on/off gravity wave drag scheme	`.true.` for all spectral resolutions, except T21 for which it is `.false.`

table continued on next page

Table 2.10: `physctl` — continued

`lice`	logical	switches on/off sea–ice temperature calculation	`.true.`
`lice_supersat`	logical	switches on/off saturation over ice for cirrus clouds (former `icnc` = 2)	`.false.`
`lmfpen`	logical	switches on/off penetrative convection	`.true.`
`lphys`	logical	switches on/off the parameterisation of diabatic processes	`.true.`
`lrad`	logical	switches on/off radiation calculation	`.true.`
`lsurf`	logical	switches on/off surface–atmosphere exchanges	`.true.`
`lvdiff`	logical	switches on/off vertical diffusion processes	`.true.`
`nauto`	integer	autoconversion scheme (not yet implemented)	`1`
`ncd_activ`	integer	type of cloud droplet activation scheme (not yet implemented)	`0`
`ncvmicro`	integer	microphysics scheme in convection parameterization (not yet implemented)	`0`

2.2.1.13 Namelist `radctl`

The namelist `radctl` controls the radiation calculation, in particular the frequency of the calls of the full radiation scheme, and which greenhouse gas concentrations and aerosol properties are taken into account. See the scientific documentation of ECHAM6 for futher details. For some namelist variables, special documentation exists and can be provided by S. Rast (sebastian.rast@zmaw.de): 3d-ozone climatology (cr2010_04_08), CO_2 submodel (cr2009_12_10), stratospheric aerosols by T. Crowley or HAM (cr2011_03_23), tropospheric aerosols by S. Kinne (cr2009_01_09), variable solar irradiance (cr2010_04_01), volcanic aerosols by G. Stenchikov (cr2010_03_15).

Table 2.11: Namelist `radctl`

Variable	type	Explanation	default
`cecc`	double prec	eccentricity of the orbit of the earth	`0.016715`
`cfcvmr(1:2)`	double prec	CFC volume mixing ratios for CFC11 and CFC12 if `icfc=2`	252.8×10^{-12}, 466.2×10^{-12}
`ch4vmr`	double prec	CH_4 volume mixing ratio (mole fraction) for `ich4=2,3`	1693.6×10^{-9}
`clonp`	double prec	longitude of perihelion measured from vernal equinox in degrees	`282.7`
`co2vmr`	double prec	CO_2 volume mixing ratio (mole fraction) for `ico2=2`	353.9×10^{-06}
`cobld`	double prec	obliquity of the orbit of the earth in degrees	`23.441`

table continued on next page

Table 2.11: `radctl` — continued

`fco2`	double prec	if an external co2 scenario (`ighg` = `1` and `ico2` = `4`) is used, the CO_2 concentrations are multiplied by `fco2`	`1.`
`iaero`	integer	`iaero` = 0: the aerosol concentrations are set to zero in the radiation computation `iaero` = 1: prognostic aerosol of a submodel (HAM) `iaero` = 2: climatological Tanre aerosols `iaero` = 3: aerosol climatology compiled by S. Kinne `iaero` = 5: aerosol climatology compiled by S. Kinne complemented with the volcanic aerosols of G. Stenchikov `iaero` = 6: aerosol climatology compiled by S. Kinne complemented with the volcanic aerosols of G. Stenchikov plus additional (stratospheric) aerosols from submodels like HAM. The additional aerosol optical properties are computed from effective radii and the aerosol optical depth at 550 nm, both quantities provided by external files with the help of a lookup table by S. Kinne (b30w120), see Tab. 2.28 `iaero` = 7: aerosol climatology compiled by S. Kinne complemented by the volcanic aerosols by T. Crowley that are computed using the lookup table by S. Kinne (b20w120), see Tab. 2.28 There is no `iaero` = 4.	2
`icfc`	integer	`icfc` = 0: all chloro–fluoro–carbon (CFC) concentrations are set to zero for the radiation computation `icfc` = 1: transported CFCs by any submodel (not yet implemented) `icfc` = 2: uniform volume mixing ratios as defined in the 2–element vector `cfcvmr(1:2)` are used for CFC11 and CFC12, respectively `icfc` = 4: uniform volume mixing ratios for a specific scenario defined by `ighg` are used in the radiation computation	2

table continued on next page

Table 2.11: `radctl` — continued

`ich4`	integer	`ich4` = 0: CH_4 concentration is set to zero for the radiation computation `ich4` = 1: transported CH_4 by any submodel (not yet implemented) `ich4` = 2: uniform volume mixing ratio `ch4vmr` of methane used in radiation computation `ich4` = 3: in the troposphere a volume mixing ratio `ch4vmr` with a decay in the layers above the troposphere is used in the radiation compution `ich4` = 4: a uniform volume mixing ratio for a certain scenario defined by the parameter `ighg` is used in the radiation computation	3
`ico2`	integer	`ico2` = 0: CO_2 concentration set to zero for the radiation computation `ico2` = 1: interactively calculated CO_2 volume mixing ratio is used with a start value of `co2vmr` `ico2` = 2: uniform volume mixing ratio `co2vmr` used in radiation computation `ico2` = 4: uniform volume mixing ratio for a certain scenario run defined by the `ighg` parameter is used	2
`ighg`	integer	`ighg` = 0: no specific scenario is chosen `ighg` = 1: a certain scenario of greenhouse gas volume mixing ratios is used. **Caution**: the variables `icfc`, `ich4`, `ico2`, `in2o` have to be set to the values corresponding to the usage of a scenario in that case	
`ih2o`	integer	`ih2o` = 0: H_2O is not taken into account in the radiation computation, i.e. specific humidity, cloud water, cloud ice are all set to zero for the radiation computation `ih2o` = 1: use prognostic specific humidity, cloud water and cloud ice in radiation computation	1

table continued on next page

Table 2.11: `radctl` — continued

`in2o`	integer	`in2o` = 0 : the N_2O concentration is set to zero for the radiation computation `in2o` = 1: transported N_2O by any submodel (not yet implemented) `in2o` = 2: a uniform volume mixing ratio of `n2ovmr` is used for the radiation computation `in2o` = 3: a uniform volume mixing ratio of `n2ovmr` is used in the troposphere with a decay in the layers above the troposphere for the radiation computation `in2o` = 4: a uniform volume mixing ratio of N_2O for a specific scenario run defined by `ighg` is used for the radiation computation	3
`io3`	integer	`io3` = 0: the O_3 concentration is set to zero for the radiation computation `io3` = 1: transported O_3 by any submodel (not yet implemented) `io3` = 2: climatological O_3 volume mixing ratios given in spectral space are used in the radiation computation as it was done in ECHAM4 `io3` = 3: climatological O_3 volume mixing ratios given in gridpoint space in a NetCDF file are used in the radiation computation `io3` = 4: climatological O_3 volume mixing ratios provided by the IPCC process in NetCDF files are used for the radiation calculation	3
`io2`	integer	`io2` = 0: the O_2 concentration is set to zero for the radiation computation `io2` = 2: the O_2 volume mixing ratio is set to `o2vmr` for the radiation computation.	2

table continued on next page

Table 2.11: `radctl` — continued

`isolrad`	integer	controls choice of solar constant. `isolrad` = 0: standard rrtm solar constant `isolrad` = 1: time dependent spectrally resolved solar constant read from file `isolrad` = 2: pre–industrial solar constant `isolrad` = 3: solar constant for amip runs (fixed in time)	3
`l_lrtm`	logical	switches on/off new LRTM radiation scheme	`.true`
`l_newoptics`	logical	switches on/off new optical parameters of clouds	`.true.`
`l_srtm`	logical	switches on/off new RRTM solar radiation scheme	`.true`
`ldiur`	logical	switches on/off diurnal cycle	`.true.`
`lradforcing(2)`	logical	switches on/off the diagnostic of instantaneous aerosol forcing in the solar spectral range (`lradforcing(1)`) and the thermal spectral range (`lradforcing(2)`).	`.false.,.false.`
`n2ovmr`	double prec	N_2O volume mixing ratio (mole fraction) for `in2o=2,3`	309.5×10^{-9}
`nmonth`	integer	`nmonth` = 0: execute full annual cycles `nmonth` = $1, 2, \ldots, 12$: perpetual repetition of the month corresponding to the number to which `nmonth` is set. The perpetual month works with a 360–day orbit only (`l_orbvsop87=.false.` must be set in `runctl`).	0
`o2vmr`	double prec	O_2 volume mixing ratio	`0.20946`
`trigrad`	special	time interval for radiation calculation	`2,'hours','first',0`
`yr_perp`	integer	year in the Julian calendar for perpetual year simulations. Works with `l_orbvsop87=.true.` only.	`-99999`

2.2.1.14 Namelist `runctl`

This namelist contains variables which control the start and end of a simulation and general properties of the output. For some namelist variables, special documentation exists and can be provided by S. Rast (sebastian.rast@zmaw.de): debug stream (cr2009_03_31) and tendency diagnostic (cr2011_01_18).

Table 2.12: Namelist `runctl`

Variable	type	Explanation	default
`delta_time`	integer	time step length in seconds	default depends on model resolution, e.g.: T63L47: 600 s, T63L95: 450 s, T127L95: 240 s
`dt_resume`	integer	reset restart date to a user defined value. Is of the form yy,mo,dy,hr,mi,se (year, month, day, hour, minute, second)	`0,0,0,0,0,0`
`dt_start(1:6)`	integer	vector of 6 integer numbers defining the start date of the experiment of the form yy,mo,dy,hr,mi,se (year, month, day, hour, minute, second)	`0,0,0,0,0,0`
`dt_stop`	integer	stop date of experiment. Is of the form yy,mo,dy,hr,mi,se (year, month, day, hour, minute, second)	`0,0,0,0,0,0`
`gethd`	special	time interval for getting data from hydrological discharge model	`1,'days','off',0`
`getocean`	special	time interval for sending atmospheric data to an ocean program coupled to ECHAM5	`1,'days','off',0`
`iadvec`	integer	selection of the advection scheme: `iadvec` = 0: no advection of trace species and water vapour `iadvec` = 1: semi Lagrangian transport algorithm `iadvec` = 2: spitfire advection scheme `iadvec` = 3: flux form semi Lagrangian transport (Lin and Rood)	`3` – flux form semi Lagrangian transport
`l_orbvsop87`	logical	`l_orbvsop87` = `.true.`: use orbit functions from vsop87 (real orbit); `l_orbvsop87` = `.false.`: "climatological" pcmdi (AMIP) orbit	`.true.`
`l_volc`	logical	switches on/off volcanic aerosols. This variable is obsolete and has to be removed. Use `iaero` of the `radctl` namelist instead.	`.false.`
`lamip`	logical	switches on/off the use of a timeseries of sea surface temperatures (AMIP style simulation)	`.false.`
`lcollective_write`	logical	switch on/off parallel writing of restart files	`.false.`

table continued on next page

Table 2.12: `runctl` — continued

`lcouple`	logical	switches on/off coupling with ocean	`.false.`
`lcouple_co2`	logical	switches on/off the interactive CO_2 budget calculation in a coupled atmosphere/ocean run	`.false.`
`ldailysst`	logical	switches on/off daily varying sea surface temperature and sea ice	`.false.`
`ldebug`	logical	switches on/off mass fixer diagnostics	`.false.`
`ldebugev`	logical	switches on/off the output of debugging information about events	`.false.`
`ldebughd`	logical	switches on/off the output of debugging information about the hydrological discharge model	`.false.`
`ldebugio`	logical	switches on/off the output of debugging information about input and output	`.false.`
`ldebugmem`	logical	switches on/off the output of debugging information about memory use	`.false.`
`ldebugs`	logical	switches on/off the debug stream	`.false.`
`ldiagamip`	logical	switches on/off AMIP diagnostics	`.false.`
`lhd`	logical	switches on/off the coupling to the hydrologic discharge model (HD model)	`.false.`
`lhd_highres`	logical	switches on/off high resolution (0.5°) output of hydrological discharge model	`.false.`
`lhd_que`	logical	switches on/off additional output from hydrological discharge model	`.false.`
`lindependent_read`	logical	switches on/off reading initial or restart data by each MPI rank separately	`.false.`
`lipcc`	logical	switches on/off the use of IPCC parameters	`.false.`
`lmeltpond`	logical	switches on/off the presence of meltponds in albedo calculation	`.true.`
`lmidatm`	logical	switches on/off middle atmosphere model version	`.true.`
`lmlo`	locical	switches on/off mixed layer ocean	`.false.`
`lnmi`	logical	switches on/off normal mode initialisation	`.false.`
`lnudge`	logical	switches on/off the "nudging" i.e. constraining the dynamic variables divergence, vorticity, temperature, and surface pressure towards given external values by relaxation	`.false.`
`lnwp`	logical	switches on/off Numerical Weather Prediction mode	`.false.`
`lport`	logical	switches on/off the introduction of a random perturbation for portability tests	`.false.`

table continued on next page

Table 2.12: `runctl` — continued

`lprint_m0`	logical	switches on/off measuring and printing the cpu time for every time step	`.false.`
`lresume`	logical	`lresume` = `.true.`: perform a rerun `lresume` = `.false.`: perform an initial run	`.false.`
`lroot_io`	logical	disables (`.true.`) or enables (`.false.`) classical root I/O.	`.true.`
`ltctest`	logical	switches on/off a test of time control without performing a true simulation	`.false.`
`ltdiag`	logical	switches on/off an additional detailed tendency diagnostic	`.false.`
`ltimer`	logical	switches on/off the output of some performance related information (run time)	`.false.`
`ly360`	logical	switches on/off the use of a 360–day year	`.false.`
`ndiahdf`	integer	logical unit number for file containing horizontal diffusion diagnostics.	`10`
`nhd_diag`	integer	number of region for which hydrological discharge model diagnostics is required	`0`
`no_cycles`	integer	stop after `no_cycles` of reruns	`1`
`no_days`	integer	stop after `no_days` days after `dt_start`	`-1`
`no_steps`	integer	stop after the integration of `no_steps` of time steps after `dt_start`	`-1`
`nproma`	integer	vector length of calculations in grid point space	number of longitudes
`nsub`	integer	number of subjobs	`0`
`out_datapath`(256)	character	name of path to which output files are written. Must have a "/" at the end	`' '`
`out_expname`(19)	character	prefix of output file names	`' '`
`out_filetype`	integer	format of meteorological output files `out_filetype` = 1: GRIB format `out_filetype` = 2: NetCDF format `out_filetype` = 6: NetCDF4 format	`1`
`out_ztype`	integer	compression type of outputfiles `out_ztype` = 0: no compression `out_ztype` = 1: szip only for GRIB output `out_ztype` = 2: zip only for NetCDF4 output	`0`
`putdata`	special	time interval at which output data are written to output files	`12,'hours','first',0`
`puthd`	special	time interval for putting data to the hydrological discharge model	`1,'days','off',0`

table continued on next page

Table 2.12: `runctl` — continued

`putocean`	special	time interval for receiving ocean data in the atmospheric part if ECHAM6 is coupled to an ocean model	`1,'days','off',0`
`putrerun`	special	time interval for writing rerun files	`1,'months','last',0`
`rerun_filetype`	integer	format of rerun files `rerun_filetype` = 2: NetCDF format `rerun_filetype` = 4: NetCDF2 format	`2`
`subflag(1:9)`	logical	vector of nine switches for switching on/off the binding of subjob output to output streams	`.false.`
`trac_filetype`	integer	format of tracer output files `trac_filetype` = 1: GRIB format `trac_filetype` = 2: NetCDF format	`1`
`trigfiles`	special	time interval at which new output files are opened	`1,'months','first',0`
`trigjob`	special	time interval for the automatic submission of subjobs	`1,'months','off',0`

2.2.1.15 Namelist `submdiagctl`

This namelist controls diagnostic output of generic submodel variables and streams. In the "pure" ECHAM6 version, these switches do not have any functionality.

Table 2.13: Namelist `submdiagctl`

Variable	type	Explanation	default
`drydep_gastrac(24,1:200)`	character	names of gas phase tracers to be included in dry deposition stream. Special name `'default'` is possible	`drydep_gastrac(1) = 'default', drydep_gastrac (2:200) = ''`
`drydep_keytype`	integer	aggregation level of output of dry deposition stream `drydep_keytype=1`: output by tracer `drydep_keytype=2`: output by (chemical) species `drydep_keytype=3`: output by (aerosol) mode `drydep_keytype=4`: user defined	`2`
`drydep_lpost`	logical	switches on/off output of wet deposition stream	`.true.`
`drydep_tinterval`	special	output frequency of wet deposition stream	`putdata` (see `runctl` namelist)

table continued on next page

Table 2.13: `submdiagctl` — continued

`drydepnam(32,1:50)`	character	list of tracer names of dry deposition output stream. There are the special names `'all' = 'detail'`, and `'default'`	`drydepnam(1) = 'default'`, `drydepnam(2:50) = ''`
`sedi_keytype`	integer	aggregation level of output of sedimentation stream `sedi_keytype=1`: output by tracer `sedi_keytype=2`: output by (chemical) species `sedi_keytype=3`: output by (aerosol) mode `sedi_keytype=4`: user defined	2
`emi_gastrac(24,1:200)`	character	names of gas phase tracers to be included in emission stream to diagnose emissions. Special name `'default'` is possible	`emi_gastrac(1) = 'default'`, `emi_gastrac (2:200) = ''`
`emi_keytype`	integer	aggregation level of output of emission diagnostic stream `emi_keytype=1`: output by tracer `emi_keytype=2`: output by (chemical) species `emi_keytype=3`: output by (aerosol) mode `emi_keytype=4`: user defined	2
`emi_lpost`	logical	switches on/off output of emission diagnostic stream	`.true.`
`emi_lpost_detail`	logical	switches on/off detailed (emissions by sector) output of emission diagnostic stream	`.true.`
`emi_tinterval`	special	output frequency of emission diagnostic stream	`putdata` (see `runctl` namelist)
`eminam(32,1:50)`	character	list of tracer names of emission diagnostic stream. There are the special names `'all' = 'detail'`, and `'default'`	`eminam(1) = 'default'`, `eminam(2:50) = ''`

table continued on next page

Table 2.13: `submdiagctl` — continued

`sedi_keytype`	integer	aggregation level of output of sedimentation stream `sedi_keytype=1`: output by tracer `sedi_keytype=2`: output by (chemical) species `sedi_keytype=3`: output by (aerosol) mode `sedi_keytype=4`: user defined	`2`
`sedi_lpost`	logical	switches on/off output of sedimentation stream	`.true.`
`sedi_tinterval`	special	output frequency of sedimentation stream	`putdata` (see `runctl` namelist)
`sedinam(32,1:50)`	character	list of tracer names of sedimentation diagnostic stream. There are the special names $'$`all`$'$ = $'$`detail`$'$, and `'default'`	`sedinam(1) = 'default', sedinam(2:50) = ''`
`vphysc_lpost`	logical	switches on/off output of vphysc stream	`.true.`
`vphyscnam(32,1:50)`	character	list of variable names of vphysc stream. There are the special names `'all'` and `'default'`	`vphyscnam(1) = 'default', vphyscnam(2:50) = ''`
`vphysc_tinterval`	special	output frequency of vphysc stream	`putdata` (see `runctl` namelist)
`wetdep_gastrac(24,1:200)`	character	names of gas phase tracers to be included in wet deposition stream. Special name `'default'` is possible	`wetdep_gastrac(1) = 'default', wetdep_gastrac (2:200) = ''`
`wetdep_keytype`	integer	aggregation level of output of wet deposition stream `wetdep_keytype=1`: output by tracer `wetdep_keytype=2`: output by (chemical) species `wetdep_keytype=3`: output by (aerosol) mode `wetdep_keytype=4`: user defined	`2`
`wetdep_lpost`	logical	switches on/off output of wet deposition stream	`.true.`

table continued on next page

Table 2.13: `submdiagctl` — continued

`wetdep_tinterval`	special	output frequency of wet deposition stream	`putdata` (see `runctl` namelist)
`wetdepnam(32,1:50)`	character	list of tracer names of wet deposition output stream. There are the special names `'all'` = `'detail'`, and `'default'`	`wetdepnam(1) = 'default'`, `wetdepnam(2:50) = ''`

2.2.1.16 Namelist `submodelctl`

This namelist contains general submodel switches of "proper submodels" including switches that control the coupling among submodels.

Table 2.14: Namelist `submodelctl`

Variable	type	Explanation	default
`laircraft`	logical	switches on/off aircraft emissions	`.false.`
`lburden`	logical	switches on/off burden calculation (column integrals)	`.false.`
`lco2`	logical	switches on/off CO_2 submodel (interacting with JSBACH)	`.false.`
`lchemheat`	logical	switches on/off chemical heating	`.false.`
`lchemistry`	logical	switches on/off chemistry	`.false.`
`ldrydep`	logical	switches on/off dry deposition	`.false.`
`lham`	logical	switches on/off HAM aerosol submodel	`.false.`
`lemissions`	logical	switches on/off emissions	`.false.`
`lhammonia`	logical	switches on/off HAMMONIA submodel (middle and upper atmosphere submodel)	`.false.`
`lhammoz`	logical	switches on/off HAM aerosol submodel and MOZART chemistry submodel and the coupling between the two	`.false.`
`lhmzhet`	logical	switches on/off hammoz heterogeneous chemistry	`.false.`
`lhmzphoto`	logical	switches on/off hammoz photolysis	`.false.`

table continued on next page

Table 2.14: `submodelctl` — continued

`lhmzoxi`	logical	switches on/off hammoz oxidant fields	`.false.`
`linterchem`	logical	switches on/off coupling of chemistry with radiation	`.false.`
`linteram`	logical	switches on/off interactive airmass calculation (HAMMONIA)	`.false.`
`lintercp`	logical	switches on/off interactive c_p calculation (HAMMONIA)	`.false.`
`llght`	logical	switches on/off interactive computation of lightning emissions	`.false.`
`lmethox`	logical	switches on/off methane oxidation in stratosphere	`.false.`
`lmegan`	logical	switches on/off biogenic vegetation emissions	`.false.`
`lmicrophysics`	logical	switches on/off microsphysics calculations	`.false.`
`lmoz`	logical	switches on/off MOZART chemistry submodel	`.false.`
`loisccp`	logical	switches on/off isccp simulator. Currently, the isccp simulator is implemented outside the submodel interface	`.false.`
`losat`	logical	switches on/off satellite simulator. Currently, the `locosp` switch for the cosp satellite simulator is implemented outside the submodel interface.	`.false.`
`lsalsa`	logical	switches on/off SALSA aerosol submodel	`.false.`
`lsedimentation`	logical	switches on/off sedimentation	`.false.`
`ltransdiag`	logical	switches on/off atmospheric energy transport diagnostic	`.false.`
`lwetdep`	logical	switches on/off drydeposition	`.false.`
`lxt`	logical	switches on/off generic test of tracer submodel	`.false.`

2.2.1.17 Namelist `tdiagctl`

This namelist determines the output of the tendency diagnostic. The tendencies of Tab. 2.15 can be diagnosed. The following variables are contained in the diagnostic stream `tdiag`. The top row describes the variables, the first column gives the routine names (processes) producing the tendencies saved under the names in the corresponding rows. The units of the variables and code numbers are given in parenthesis.

Table 2.15: Variables of the diagnostic stream `tdiagctl`

variable / **routine (process)**	du/dt (m/s/day)	dv/dt (m/s/day)	dT/dt (K/day)	dq/dt (1/day)	dx_l/dt (1/day)	dx_i/dt (1/day)
vdiff	`dudt_vdiff` (code 11)	`dvdt_vdiff` (code 21)	`dtdt_vdiff` (code 1)	`dqdt_vdiff` (code 31)	`dxldt_vdiff` (code 41)	`dxidt_vdiff` (code 51)
radheat	—	—	`dtdt_rheat_sw` (code 62)	—	—	—
	—	—	`dtdt_rheat_lw` (code 72)	—	—	—
gwspectrum	`dudt_hines` (code 13)	`dvdt_hines` (code 23)	`dtdt_hines` (code 3)	—	—	—
ssodrag	`dudt_sso` (code 14)	`dvdt_sso` (code 24)	`dtdt_sso` (code 4)	—	—	—
cucall	`dudt_cucall` (code 15)	`dvdt_cucall` (code 25)	`dtdt_cucall` (code 5)	`dqdt_cucall` (code 35)	—	—
cloud	—	—	`dtdt_cloud` (code 6)	`dqdt_cloud` (code 36)	`dxldt_cloud` (code 46)	`dxidt_cloud` (code 56)
atmospheric variables						
Box area m^2	surface geopotential $\mathrm{m}^2/\mathrm{s}^2$		$\ln(p_\mathrm{s}/p^\ominus)$ spectral	p_s Pa	$T(t)$ spectral	$T(t-\Delta t)$ K

Additional documentation can be found in cr2011_01_18 provided by S. Rast (sebastian.rast@zmaw.de).

Table 2.16: Namelist `tdiagctl`

Variable	type	Explanation	default
`puttdiag`	special	Output frequency of tendency stream	`6, 'hours', 'first', 0`

table continued on next page

Table 2.16: `tdiagctl` — continued

`tdiagnam(32,1:22)`	character	determines the choice of tendencies that are written to the output stream `_tdiag` keyword: explanation `'all'`: output all tendencies of `tdiag` stream one of `'vdiff'`, `'radheat'`, `'gwspectrum'`, `'ssodrag'`, `'cucall'`, `'cloud'`: output all tendencies associated with **vdiff**, **radheat**, **gwspectrum**, **ssodrag**, **cucall**, **cloud** one of `'uwind'`, `'vwind'`, `'temp'`, `'qhum'`, `'xl'`, `'xi'`: of all processes, output the tendency du/dt, dv/dt, dT/dt, dq/dt, dx_l, dx_i one of the variable names of the tendencies listed in table 2.15, e.g. `dudt_hines`: output this tendency, e.g. du/dt due to `gwspectrum`	`tdiagnam`(1) = 'all', `tdiagnam`(2 : 22) = 'end'

2.2.2 Input namelists in file `namelist.jsbach`

The JSBACH namelist file `namelist.jsbach` contains several independent Fortran namelists:

`albedo_ctl`: defines parameters that are used in the albedo scheme

`bethy_ctl`: controls the photosynthesis (BETHY) module

`cbalance_ctl`: defines parameters of the carbon module

`climbuf_ctl`: defines parameters for multi–year climate variable calculation

`dynveg_ctl`: controls the dynamic vegetation

`jsbach_ctl`: defines the basic settings of a JSBACH simulation. The namelist includes parameters to switch on or off JSBACH modules, and controls IO.

`soil_ctl`: defines parameters used in the soil module

The tables in the following subsections list all namelist parameters of the different JSBACH namelists. Each parameter is listed in alphabetical order and is briefly described. Besides, the Fortran type and the default values are given.

2.2.3 Namelist `albedo_ctl`

The namelist for the albedo scheme is read in routine `config_albedo` of module `mo_land_surface.f90`. It is used only if the albedo scheme is switched on, i.e. `use_albedo=.TRUE.` in namelist `jsbach_ctl` (compare table 2.22).

Table 2.17: Namelist `albedo_ctl`

Parameter	Type	Description	Default
`use_albedocanopy`	logical	`.TRUE.`: read maps of canopy albedo (`albedo_veg_nir` and `albedo_veg_vis` from `jsbach.nc`); `.FALSE.`: use PFT specific albedo values from `lctlib.def`	`.FALSE.`
`use_snowage`	logical	if `.TRUE.`, account for snow aging in albedo calculation	`.TRUE.`

2.2.4 Namelist `bethy_ctl`

The namelist `bethy_ctl` controls the BETHY module for photosynthesis. It is used only if `use_bethy=.TRUE.` in namelist `jsbach_ctl` (compare table 2.22). The namelist is read in routine `config_bethy` of `mo_bethy.f90`.

Table 2.18: Namelist `bethy_ctl`

Parameter	Type	Description	Default
`ncanopy`	integer	number of canopy layers	3

2.2.5 Namelist `cbalance_ctl`

The cbalance module handling the carbon pools is controlled by namelist `cbalance_ctl`. The namelist is read in routine `init_cbalance_bethy` in `mo_cbal_bethy.f90`.

Table 2.19: Namelist `cbalance_ctl`

Parameter	Type	Description	Default

table continued on next page

Table 2.19: `cbalance_ctl` — continued

`cpools_file_name`	character	name of the file containing initial data for the carbon pools. Only used if `read_cpools=.TRUE.`	`'Cpools.nc'`
`ndepo_file_name`	character	name of the file containing nitrogen deposition data. Only used if `with_nitrogen=.TRUE.` in `jsbach_ctl` and `read_cpools=.TRUE.`	`'Ndepo.nc'`
`npools_file_name`	character	name of the file containing initial data for the nitrogen pools. Only used if `with_nitrogen=.TRUE.` in `jsbach_ctl` and `read_npools=.TRUE.`	`'Npools.nc'`
`read_cpools`	logical	initialize carbon pools with data from an external file.	`.FALSE.`
`read_ndepo`	logical	read nitrogen deposition data from an external file. Only used if `with_nitrogen=.TRUE.` in `jsbach_ctl`	`.FALSE.`
`read_npools`	logical	initialize nitrogen pools with data from an external file. Only used if `with_nitrogen=.TRUE.` in `jsbach_ctl`	`.FALSE.`

2.2.6 Namelist `climbuf_ctl`

The climate buffer provides climate variables as multi-annual running means, minimums or maximums. It is controlled by namelist `climbuf_ctl`. The namelist is read in routine `config_climbuf` (`mo_climbuf.f90`).

Table 2.20: Namelist `climbuf_ctl`

Parameter	Type	Description	Default
`init_running_means`	logical	initialize the calculation of long term climate variables. (Should be `.TRUE.` at the beginning of the second year of an initialized experiment.)	`.FALSE.`
`read_climbuf`	logical	read climate buffer data from an external file.	`.FALSE.`
`climbuf_file_name`	character	name of the climate buffer file. Only used if `read_climbuf=.TRUE.`	`'climbuf.nc'`

2.2.7 Namelist `dynveg_ctl`

The dynamic vegetation is controlled by `dynveg_ctl`. The namelist is read in `config_dynveg` (`mo_dynveg.f90`). It is used only, if the dynamic vegetation is switched on by setting `USE_DYNVEG= .TRUE.` in namelist `jsbach_ctl` (compare table 2.22).

Table 2.21: Namelist `dynveg_ctl`

Parameter	Type	Description	Default
`accelerate_dynveg`	real	factor to accelerate vegetation dynamics. Default: no acceleration	1.
`dynveg_all`	logical	activate competition between woody types and grasses (not recommended)	`.FALSE.`
`dynveg_feedback`	logical	switch on/off the feedback of the dynamic vegetation on the JSBACH physics. (Cover fractions are kept constant, while fire and wind break still influence the carbon cycle.)	`.TRUE.`
`fpc_file_name`	character	name of an external vegetation file. Only used if `read_fpc=.TRUE.`	`'fpc.nc'`
`read_fpc`	logical	read initial cover fractions from an external file; the file name is defined with parameter `fpc_file_name`.	`.FALSE.`

2.2.8 Namelist `jsbach_ctl`

The namelist `jsbach_ctl` includes the basic parameters for a JSBACH simulation. It is needed to switch on or off the different physical modules as e.g. the dynamic vegetation or the albedo scheme. Besides, it controls file names and other IO-options. The namelist is read in routine `jsbach_config` of module `mo_jsbach.f90`.

Table 2.22: Namelist `jsbach_ctl`

Parameter	Type	Description	Default
`debug`	logical	additional output for debugging	`.FALSE.`
`debug_Cconservation`	logical	additional debugging output to solve problems with carbon conservation	`.FALSE.`
`file_type`	character	output format: `GRIB`, `NETCDF`, `NETCDF2` or `NETCDF4`	`'GRIB'`
`grid_file`	character	input file containing grid information	`'jsbach.nc'`
`lcc_forcing_type`	character	Scheme for (anthropogenic) landcover changes. `NONE`: no landcover change; `MAPS`: read maps of landcover fractions; `TRANSITIONS`: read maps with landuse transitions	`'NONE'`
`lctlib_file`	character	name of the land cover library file	`'lctlib.def'`
`lpost_echam`	logical	if `.TRUE.`, write jsbach output variables, even if they are part of the echam output	`.FALSE.`

table continued on next page

Table 2.22: `jsbach_ctl` — continued

`c5compat`	logical	if `.TRUE.`, preserve compatibility in JSBACH carbon handling with CMIP5 simulations (root exudates to litter pools)	`.TRUE.`
`missing_value`	real	missing value for the output (ocean values)	`NF_FILL_REAL`
`ntiles`	integer	number of tiles defined on each grid cell	`-1`
`read_cover_fract`	logical	read cover fractions from the JSBACH initial file rather than from restart file	`.FALSE.`
`soil_file`	character	file containing initial data of soil properties	`'jsbach.nc'`
`standalone`	logical	Type of model run; `.TRUE.`: standalone JSBACH run; `.FALSE.`: JSBACH driven by an atmosphere model	`.TRUE.`
`surf_file`	character	file containing initial data of the land surface	`'jsbach.nc'`
`test_Cconservation`	logical	switches on/off carbon conservation test	`.FALSE.`
`test_stream`	logical	additional stream for model testing	`.FALSE.`
`use_albedo`	logical	switches on/off a dynamic albedo scheme	`.FALSE.`
`use_bethy`	logical	switches on/off the BETHY model (photosynthesis, respiration)	`.FALSE.`
`use_dynveg`	logical	switches on/off the dynamic vegetation module	`.FALSE.`
`use_phenology`	logical	switches on/off the phenology module to calculate the LAI	`.FALSE.`
`veg_file`	character	file containing initial data for the vegetation	`'jsbach.nc'`
`with_nitrogen`	logical	calculate the nitrogen cycle (not fully implemented in the current version).	`.FALSE.`

2.2.9 Namelist `soil_ctl`

The configurable parameters to control the soil physics are defined in namelist `soil_ctl`. The namelist is read in `config_soil` in module `mo_soil.f90`.

Table 2.23: Namelist `soil_ctl`

Parameter	Type	Description	Default
`crit_snow_depth`	real	Critical snow depth for correction of surface temperature for melting [m]	5.85036×10^{-03}

table continued on next page

Table 2.23: `soil_ctl` — continued

`moist_crit_fract`	real	critical value of soil moisture above which transpiration is not affected by the soil moisture stress; expressed as fraction of the maximum soil moisture content	`0.75`
`moist_max_limit`	real	upper limit for maximum soil moisture content: If positive, `max_moisture` from initial file is cut off at this value.	`-1.0`
`moist_wilt_fract`	real	soil moisture content at permanent wilting point, expressed as fraction of maximum soil moisture content	`0.35`
`skin_res_max`	real	maximum water content of the skin reservoir of bare soil [m]	$2. \times 10^{-04}$

2.2.10 Input namelists in other files

2.2.10.1 Namelist `mvctl`

For each stream in the `mvstreamctl` namelist, a `mvctl` namelist has to be created. The `mvctl` namelist has to be written to a file {`namelist`}`.nml` where {`namelist`} is the name of the respective stream. For tracers, the namelist has to be written to `tracer.nml`. See section 2.2.1.8 also. Additional documentation can be found in cr2010_07_28 provided by S. Rast (sebastian.rast@zmaw.de).

Table 2.24: Namelist `mvctl`

Variable	type	Explanation	default
`putmean`	special	frequency at which the respective mean value stream shall be written	1,'months','first',0
`meannam(32,1:500)`	character	list of variables (e.g. meteorological variables, chemical species) for which mean values are calculated	empty strings

table continued on next page

Table 2.24: `mvctl` — continued

`stddev(1:500)`	integer	This variable controls the calculation of the mean of the square of each variable in the list `meannam`. `stddev(1)` = −1: calculate the mean square of all variables present in `meannam` `stddev(i)` = 0: Do not calculate the mean of the square of variable `i` except if `stddev(1)` = −1 `stddev(i)` = 1: Calculate the mean of the square of variable `i` in list `meannam`.	`0,0,...,0`

2.3 Input data

This section provides a brief description of the input files but does not describe the input data itself. Such a description can be found in the scientific part of the documentation or (at least in parts) on the web: :
`http://www.mpimet.mpg.de/en/science/models/echam/echam5/inputoutput/echam5-input-files.html`
All input files are stored in the directory
`/pool/data/ECHAM6/`
and its subdirectories for the atmospheric part and in the directory
`/pool/data/JSBACH`
for the land–surface model. In `/pool/data/ECHAM6/`, you find the resolution independent data. Furthermore, it contains directories {`RES`} where {`RES`} has to be replaced by one of the spectral model resolutions T31, T63, T127, and T255, respectively providing resolution dependent input files. Similarly, the resolution dependent land–surface model data are stored in subdirectories `T31`, `T63`, etc. of the `/pool/data/JSBACH` directory. In the following, the vertical resolution will be denoted by {`LEV`} which represents the number of vertical σ–levels preceeded by a capital `L`. The most common model resolutions are T63L47 and T127L95. Currently, ECHAM6 is tuned for the resolutions T63L47, T63L95, T127L95 only. Other resolutions may require a new tuning of the model in order to adjust the parameters of certain equations to the particular model resolution. Some of the input data contain information about the land–sea distribution and therefore are provided for various ocean resolutions even if the model is not coupled to an interactive ocean. The ocean resolution will be symbolized by {`OCR`}. Currently, the GR15, GR30, TP04, TP10, TP6M ocean resolutions are considered, but not all possible combinations with spectral ECHAM6 resolutions are available.
There are three kinds of input data: initial conditions, boundary conditions, and data of model parameters. The boundary conditions can be either "transient boundary conditions" depending on the actual year or "climatological boundary conditions" which do not depend on the year but may contain a seasonal cycle. The files containing the initial conditions are listed in Tab. 2.25.

Table 2.25: Initial conditions for ECHAM6

Link target	Link name	Explanation
Resolution dependent ECHAM6 initial data in /pool/data/ECHAM6//{RES}		
{RES}{LEV}_jan_spec.nc	unit.23	Variables describing the vertical σ–coordinates, spectral fields like divergence, vorticity etc. serving to start the model from some initial values. These values are very rough estimates only and do not describe any dynamic state of the atmosphere that occurs with high probability!
{RES}{OCR}_jan_surf.nc	unit.24	Surface fields like land sea mask, glacier mask etc. for a start of the model from initial values.
Resolution independent ECHAM6 initial data in /pool/data/ECHAM6/		
hdstart.nc	hdstart.nc	Initial data for hydrological discharge model.

The climatological boundary condition files are listed in Tab. 2.26. Sea surface temperature and sea ice cover climatologies for ECHAM6 are based on 500 year–climatologies of our coupled control simulations and are available for the T63 resolutions only. Furthermore, some of the data are formally read by ECHAM6 but not used: The leaf area index, vegetation ratio, and albedo e.g. are calculated by the surface model JSBACH and it is impossible to use climatological values read from files. Actually, JSBACH reads these quantities again, but discards them also. Even if dynamic vegetation is switched off: This just means that the geographical distribution of vegetation types is fixed in time, but the leaf area index changes with season and soil moisture and consequently also the albedo varies with time according to the vegetation model used in JSBACH.
The input data for the hydrological discharge model (see Tab. 2.25 and Tab. 2.26) are not entirely resolution independent, but the current data can be used for a wide range of resolutions (probably not for T255?).

Table 2.26: Climatological boundary conditions for ECHAM6. Some of the climatological boundary conditions have to be linked to year dependent files. The year is symbolized by yyyy.

Link target	Link name	Explanation
Resolution dependent data in /pool/data/ECHAM6/{RES}		
{RES}_O3clim2.nc	unit.21	Zonal mean ozone climatology for radiation calculation. These files should not be used in ECHAM6, and are obsolete.

table continued on next page

Table 2.26: Climatological boundary conditions for ECHAM6 continued

{RES}_ozone_CMIP5_y1-y2.nc	ozonyyyy	3–d ozone climatology being a mean value over the years y1 to y2. Currently, y1-y2=1850-1860 and 1979-1988 is available. These files have to be linked to filenames ozonyyyy where yyyy is the actually simulated year.
{RES}{OCR}_VLTCLIM.nc	unit.90	Climatological leaf area index (monthly data).
{RES}{OCR}_VGRATCLIM.nc	unit.91	Climatological vegetation ratio (monthly data).
{RES}_TSLCLIM2.nc	unit.92	Climatological land surface temperature (monthly data).
T{RES}{OCR}_piControl-LR_sst_1880-2379.nc	unit.20	Climatological sea surface temperatures (monthly data, only in T63GR15 available).
{RES}{OCR}_piControl-LR_sic_1880-2379.nc	unit.96	Climatological sea ice data (monthly data, only in T63GR15 available).
Tropospheric aerosols		
aero2/{RES}_aeropt_kinne_sw_b14_coa.nc	aero_coarse_yyyy.nc	Optical properties of coarse mode aerosols in the solar spectral range. Since these are mostly of natural origin, climatological boundary conditions are sufficient for historic times.
aero2/{RES}_aeropt_kinne_lw_b16_coa.nc	aero_farir_yyyy.nc	Aerosol optical properties in the thermal spectral range. Only coarse mode aerosols play a role. Since these are mostly of natural origin, climatological boundary conditions are sufficient for historic times.
Land surface model JSBACH (/pool/data/JSBACH)		
jsbach/jsbach_{RES}{OCR}_{t}_yyyy.nc	jsbach.nc	Boundary conditions for land surface model JSBACH. It also depends on the ocean resolution because the land–sea mask does. The structure of JSBACH may vary with the number of tiles, encoded in {t}=4tiles, 8tiles, 11tiles, or 12tiles. Not all combinations of resolutions are available.
Resolution independent data in /pool/data/ECHAM6//{RES}		

table continued on next page

Table 2.26: Climatological boundary conditions for ECHAM6 continued

hcpara.nc	hdpara.nc	Data for hydrological discharge model.

Furthermore, various transient boundary conditions are available which can either replace their climatological counterparts or be used as supplemental conditions. Examples for transient boundary conditions are observed sea surface temperatures and sea ice data, transient greenhouse gas concentrations or data accounting for interannual variability in solar radiation, ozone concentration or aerosol optical properties. The historical sea surface temperature (SST) and sea ice cover (SIC) data are taken from the Program for Climate Model Diagnosis and Intercomparison (PCMDI, status: November 2009). A list of possible input data can be found in Tab. 2.27.

Table 2.27: ECHAM6 transient boundary conditions. Specific years are symbolized by yyyy.

Resolution dependent data in /pool/data/ECHAM6//{RES}

Link target	Link name	Explanation
amip2/{RES}_amip2sst_yyyy.nc	sstyyyy	historical sea surface temperatures (monthly data).
amip2/{RES}_amip2sic_yyyy.nc	iceyyyy	historical sea ice data (monthly data).
Tropospheric aerosols		
aero2/{RES}_aeropt_kinne_sw_b14_fin_yyyy.nc	aero_fine_yyyy.nc	Optical properties of fine mode aerosols in the solar spectrum. These aerosols are of anthropogenic origin mainly. Therefore, they depend on the year. These are the historical data.
aero2/{RES}_aeropt_kinne_sw_b14_fin_{sc}_yyyy.nc	aero_fine_yyyy.nc	Optical properties of fine mode aerosols in the solar spectrum. These aerosols are of anthropogenic origin mainly. Therefore, they depend on the year. They are provided for different scenarios for the future ({sc}= rcp26, rcp45, rcp85).
Volcanic (stratrospheric) aerosols, Stenchikov		
volcano_aerosols/strat_aerosol_ir_T{RES}_yyyy.nc	strat_aerosol_sw_yyyy.nc	Aerosol optical properties of stratospheric aerosols of volcanic origin in the solar spectral range.
volcano_aerosols/strat_aerosol_ir_T{RES}_yyyy.nc	strat_aerosol_ir_yyyy.nc	Aerosol optical properties of stratospheric aerosols of volcanic origin in the thermal spectral range.
Volcanic (stratrospheric) aerosols, provided by HAM		

table continued on next page

Table 2.27: Transient boundary conditions continued

N.N.	aoddz_ham_yyyy.nc	Aerosol optical properties as provided by the HAM model. These data have to be used together with the b30w120 parameter file of Tab. 2.28. The aerosol type described by the HAM model has to be compatible with that of the parameter file.
Transient 3d–ozone data in /pool/data/ECHAM6/{RES}/ozone2		
{RES}_ozone_CMIP5_yyyy.nc	ozonyyyy	Historic 3d–distribution of ozone in the stratosphere and troposphere.
{RES}_ozone_CMIP5_{sc}_yyyy.nc	ozonyyyy	3d–distribution of ozone in the stratosphere and troposphere for the scenarios rcp26, rcp45, and rcp85.
Resolution independent data in /pool/data/ECHAM6/		
Volcanic (stratrospheric) aerosols, T. Crowley		
ici5d-ad800-1999.asc	aodreff_crow.dat	Stratospheric aerosol optical properties of volcanic aerosols compiled by T. Crowley. All years are in one file. The b30w120 parameter file of Tab. 2.28 has to be used together with these data.
Transient solar irradiance in /pool/data/ECHAM6/solar_irradiance		
swflux_14band_yyyy.nc	swflux_yyyy.nc	Monthly spectral solar irradiance for year yyyy.
Greenhouse gas scenarios in /pool/data/ECHAM6/		
greenhouse_{sc}.nc	greenhouse_gases.nc	Transient greenhouse gas concentrations (all years in one file) for the scenarios {sc}= rcp26, rcp45, rcp85. The rcp45–file contains the historic data also.

Some of the equations used in ECHAM6 need tables of parameters. E.g. the radiation needs temperature and pressure (concentration) dependent absorption coefficients, the calculation of the aerosol optical properties at all wave lengths from the effective aerosol radius and the aerosol optical depth at a certain wavelength needs conversion factors. The surface model JSBACH needs further input parameters that are provided in a kind of a standard input file. A list of the input files containing model parameters is provided in Tab. 2.28.

Table 2.28: Input files for ECHAM6 containing parameters for various physical processes in /pool/data/ECHAM6/

Link target	Link name	Explanation

table continued on next page

Table 2.28: Parameters files continued

surrta_data	rrtadata	Tables for RRTM radiation scheme — solar radiation.
rrtmg_lw.nc	rrtmg_lw.nc	Tables for RRTMG radiation scheme — thermal radiation.
ECHAM6_CldOptProps.nc	ECHAM6_CldOptProps.nc	Optical properties of clouds.
b30w120	aero_volc_tables.dat	Parametrizations of the aerosol optical properties in the case of T. Crowley aerosols and aerosols provided by HAM. This table has to be compatible with the aerosol data.
jsbach/ lctlib_nlct21.def_rev4154	lctlib.def	Parametrization of properties of vegetation and land model JSBACH. (imported from the cosmos svn)

2.4 Output files and variables

The number and names of outputfiles depend on the model configuration. Tab. 2.29 lists all standard output files and gives an overview of the kind of variables being in these files. The names of the outputfiles are composed of the experiment name EXPNAME as it is given by the out_expname variable of the runctl namelist (see section 2.2.1.14), a date information DATE corresponding to the simulation date at which the output file was opened and an extension EXT that describes the output stream or family of output streams written to this file. GRIB format output files do not have further extensions, netcdf format output files have the additional extension .nc. The filename is therefore composed as EXPNAME_DATE_EXT[.nc].
All the variables that are written to an output file are members of so–called streams, a special data structure that allows for standardized output. Not all variables of a stream are written to output files. Detailed information about all streams and variables are written to the standard error output device when ECHAM6 is started.

Table 2.29: Output files of ECHAM6

Extension EXT	Content
cfdiag	diagnostic of 3–dimensional radiation and convective mass flux
co2	diagnostic of CO_2 submodel (carbon cycle)
cosp	COSP simulator output
echam	main echam outputfile comprising several echam streams containing 2– and 3–d atmospheric gridpoint and spectral variables
forcing	radiation fluxes and heating rates
surf	variables from the surface model JSBACH
tdiag	tendency diagnostic

table continued on next page

Table 2.29: Output files — continued

`tracer`	mass mixing ratios of (transported) trace gas species

The number of variables in each output stream also depend on the model configuration. In the case of GRIB output, information about code numbers and variables can be found in the respective files `EXPNAME_DATE_EXT.codes`. In the case of netcdf output, the explanation of the variable can be found inside the netcdf files. Some of the variables are mean values over the output interval, some are in spectral space, others in grid point space. We give tables of outputvariables of the most important output files only.

2.4.1 Output file `echam`

The `echam` output file combines the variables of several output streams (g3b, gl, and sp) and contains the main prognostic and diagonstic ECHAM6 output variables describing the dynamic state of the atmosphere.

Table 2.30: Output file `echam`. The type of the output fields can be g (instantaneous grid point variable), $\overline{\text{g}}$ (mean value over the output interval of grid point variable), s (spectral space variable). The dimension is either 2d (variable depends on longitudes and latitudes only), 3d (variable depends on longitudes, latitudes, and levels).

Name	Code	Type	Unit	Dimension	Stream	Explanation
`abso4`	235	$\overline{\text{g}}$	kg/m^2	2d	g3b	anthropogenic sulfur burden
`aclcac`	223	$\overline{\text{gl}}$	—	3d	g3b	cloud cover
`aclcov`	164	$\overline{\text{gl}}$	—	2d	g3b	total cloud cover
`ahfcon`	208	$\overline{\text{g}}$	W/m^2	2d	g3b	conductive heat flux through ice
`ahfice`	125	$\overline{\text{g}}$	W/m^2	2d	g3b	conductive heat flux
`ahfl`	147	$\overline{\text{gl}}$	W/m^2	2d	g3b	latent heat flux
`ahfliac`	110	$\overline{\text{gl}}$	W/m^2	2d	g3b	latent heat flux over ice
`ahfllac`	112	$\overline{\text{gl}}$	W/m^2	2d	g3b	latent heat flux over land
`ahflwac`	111	$\overline{\text{gl}}$	W/m^2	2d	g3b	latent heat flux over water
`ahfres`	209	$\overline{\text{gl}}$	W/m^2	2d	g3b	melting of ice
`ahfs`	146	$\overline{\text{gl}}$	W/m^2	2d	g3b	sensible heat flux
`ahfsiac`	119	$\overline{\text{gl}}$	W/m^2	2d	g3b	sensible heat flux over ice
`ahfslac`	121	$\overline{\text{gl}}$	W/m^2	2d	g3b	sensible heat flux over land
`ahfswac`	120	$\overline{\text{gl}}$	W/m^2	2d	g3b	sensible heat flux over water
`albedo`	175	$\overline{\text{g}}$	—	2d	g3b	surface albedo
`albedo_nir`	101	$\overline{\text{g}}$	—	2d	g3b	surface albedo for near infrared radiation range
`albedo_nir_dif`	82	$\overline{\text{g}}$	—	2d	g3b	surface albedo for near infrared radiation range, diffuse

table continued on next page

Table 2.30: Output file `echam` — continued

`albedo_nir_dir`	80	g	—	2d	g3b	surface albedo for near infrared radiation range, direct
`albedo_vis`	100	g	—	2d	g3b	surface albedo for visible radiation range
`albedo_vis_dif`	81	g	—	2d	g3b	surface albedo for visible radiation range, diffuse
`albedo_vis_dir`	79	g	—	2d	g3b	surface albedo for visible radiation range, direct
`alsobs`	72	g	—	2d	g3b	albedo of bare ice and snow without melt ponds
`alsoi`	122	g	—	2d	g3b	albedo of ice
`alsol`	124	g	—	2d	g3b	albedo of land
`alsom`	71	g	—	2d	g3b	albedo of melt ponds
`alsow`	123	g	—	2d	g3b	albedo of water
`ameltdepth`	77	g	m	2d	g3b	total melt pond depth
`ameltfrac`	78	g	—	2d	g3b	fractional area of melt ponds on sea ice
`amlcorac`	89	gl	W/m^2	2d	g3b	mixed layer flux correction
`ao3`	236	g	—	3d	g3b	mass mixing ratio of IPCC ozone
`apmeb`	137	gl	$kg/(m^2s)$	2d	g3b	vertical integral tendency of water
`apmegl`	221	gl	$kg/(m^2s)$	2d	g3b	P-E over land ice
`aprc`	143	gl	$kg/(m^2s)$	2d	g3b	convective precipitation
`aprl`	142	gl	$kg/(m^2s)$	2d	g3b	large scale precipitation
`aprs`	144	gl	$kg/(m^2s)$	2d	g3b	snow fall
`aps`	134	g	Pa	2d	g3b	surface pressure
`az0i`	116	g	m	2d	g3b	roughness length over ice
`az0l`	118	g	m	2d	g3b	roughness length over land
`az0w`	117	g	m	2d	g3b	roughness length over water
`barefrac`	70	g	—	2d	g3b	bare ice fraction
`dew2`	168	g	K	2d	g3b	dew point temperature at 2m above surface
`drain`	161	gl	$kg/(m^2s)$	2d	g3b	drainage
`evap`	182	gl	$kg/(m^2s)$	2d	g3b	evaporation
`evapiac`	113	gl	$kg/(m^2s)$	2d	g3b	evaporation over ice
`evaplac`	115	gl	$kg/(m^2s)$	2d	g3b	evaporation over land
`evapwac`	114	gl	$kg/(m^2s)$	2d	g3b	evaporation over water
`fage`	68	g	—	2d	g3b	aging factor of snow on ice
`friac`	97	gl	—	2d	g3b	ice cover fraction of grid box
`geosp`	129	g	m^2/s^2	2d	g3b	surface geopotential (orography)
`glac`	232	g	—	2d	g3b	fraction of land covered by glaciers

table continued on next page

Table 2.30: Output file `echam` — continued

`gld`	213	g	m	2d	g3b	glacier depth
`lsp`	152	s	—	2d	sp	nat. logarithm of surface pressure
`q`	133	g	—	3d	gl	specific humidity
`qres`	126	g	W/m^2	2d	g3b	residual heat flux for melting sea ice
`qvi`	230	gl	kg/m^2	2d	g3b	vertically integrated water vapour
`relhum`	157	g	—	3d	g3b	relative humidity
`runoff`	160	gl	$kg/(m^2s)$	2d	g3b	surface runoff and drainage
`sd`	155	s	$1/s$	3d	sp	divergence
`seaice`	210	g	—	2d	g3b	ice cover (fraction of 1-SLM)
`siced`	211	g	m	2d	g3b	ice depth
`sicepdi`	74	g	m	2d	g3b	ice thickness on melt pond
`sicepres`	76	g	W/m^2	2d	g3b	residual heat flux
`sicepdw`	73	g	m	2d	g3b	melt pond depth on sea ice
`slm`	172	g	—	2d	g3b	land sea mask (1=land, 0=sea/lake)
`sn`	141	g	m	2d	g3b	snow depth
`snacl`	222	gl	$kg/(m^2s)$	2d	g3b	snow accumulation over land
`snc`	233	g	m	2d	g3b	snow depth at the canopy
`sni`	214	g	m	2d	g3b	water equivalent of snow on ice
`snifrac`	69	g	—	2d	g3b	fraction of ice covered with snow
`snmel`	218	gl	$kg/(m^2s)$	2d	g3b	snow melt
`sofliac`	94	gl	W/m^2	2d	g3b	solar radiation energy flux over ice
`sofllac`	96	gl	W/m^2	2d	g3b	solar radiation energy flux over land
`soflwac`	95	gl	W/m^2	2d	g3b	solar radiation energy flux over water
`srad0d`	184	gl	W/m^2	2d	g3b	incoming solar radiation energy flux at top of atmosphere
`srad0u`	203	gl	W/m^2	2d	g3b	upward solar radiation energy flux at top of atmosphere
`srad0`	178	gl	W/m^2	2d	g3b	net solar radiation energy flux at top of atmosphere
`sradl`	86	gl	W/m^2	2d	g3b	solar radiation at 200 hPa
`srads`	176	gl	W/m^2	2d	g3b	net solar radiation energy flux at surface

table continued on next page

Table 2.30: Output file `echam` — continued

`sradsu`	204	g|	W/m^2	2d	g3b	upward solar radiation energy flux at surface
`sraf0`	187	g|	W/m^2	2d	g3b	net solar radiation energy flux at top of atmosphere for clear sky conditions
`srafl`	88	g|	W/m^2	2d	g3b	solar radiation energy flux at 200 hPa for clear sky conditions
`srafs`	185	g|	W/m^2	2d	g3b	net solar radiation energy flux at surface for clear sky conditions
`st`	130	s	K	3d	sp	temperature
`svo`	138	s	1/s	3d	sp	vorticity
`t2max`	201	g	K	2d	g3b	maximum temperature at 2m above surface
`t2min`	202	g	K	2d	g3b	minimum temperature at 2m above surface
`temp2`	167	g	K	2d	g3b	temperature at 2m above surface
`thvsig`	238	g	K	2d	g3b	standard deviation of virtual potential temperature at half level klevm1
`topmax`	217	g	Pa	2d	g3b	pressure of height level of convective cloud tops
`tpot`	239	g	K	3d	g3b	potential temperature
`trad0`	179	g|	W/m^2	2d	g3b	net thermal radiation energy flux at top of atmosphere
`tradl`	85	g|	W/m^2	2d	g3b	thermal radiation energy flux at 200 hPa
`trads`	177	g|	W/m^2	2d	g3b	net thermal radiation energy flux at surface
`tradsu`	205	g|	W/m^2	2d	g3b	upward thermal radiation energy flux at surface
`traf0`	188	g|	W/m^2	2d	g3b	net thermal radiation energy flux at top of atmosphere for clear sky conditions
`trafl`	87	g|	W/m^2	2d	g3b	thermal radiation energy flux at 200 hPa for clear sky conditions
`trafs`	186	g|	W/m^2	2d	g3b	thermal radiation energy flux at surface for clear sky conditions
`trfliac`	91	g|	W/m^2	2d	g3b	thermal radiation energy flux over ice

table continued on next page

Table 2.30: Output file `echam` — continued

`trfllac`	93	g̅	W/m^2	2d	g3b	thermal radiation energy flux over land
`trflwac`	92	g̅	W/m^2	2d	g3b	thermal radiation energy flux over water
`tropo`	237	g	Pa	2d	g3b	pressure of height level where tropopause is located according to WMO definition
`tsi`	102	g	K	2d	g3b	surface temperature of ice
`tsicepdi`	75	g	K	2d	g3b	ice temperature on frozen melt pond
`tslm1`	139	g	K	2d	g3b	surface temperature of land
`tsurf`	169	g̅	K	2d	g3b	surface temperature
`tsw`	103	g	K	2d	g3b	surface temperature of water
`u10`	165	g	m/s	2d	g3b	zonal wind velocity at 10m above surface
`ustr`	180	g̅	Pa	2d	g3b	zonal wind stress
`ustri`	104	g	Pa	2d	g3b	zonal wind stress over ice
`ustrl`	108	g	Pa	2d	g3b	zonal wind stress over land
`ustrw`	106	g	Pa	2d	g3b	zonal wind stress over water
`v10`	166	g	m/s	2d	g3b	meridional wind velocity at 10m above surface
`vdis`	145	g̅	W/m^2	2d	g3b	boundary layer dissipation
`vdisgw`	197	g	W/m^2	2d	g3b	gravity dissipation
`vstr`	181	g̅	Pa	2d	g3b	meridional wind stress
`vstri`	105	g	Pa	2d	g3b	meridional wind stress over ice
`vstrl`	109	g	Pa	2d	g3b	meridional wind stress over land
`vstrw`	107	g	Pa	2d	g3b	meridional wind stress over water
`wimax`	216	g	m/s	2d	g3b	maximum wind speed at 10m above surface
`wind10`	171	g	m/s	2d	g3b	wind velocity at 10m above surface
`wl`	193	g	m	2d	g3b	skin reservoir content
`ws`	140	g	m	2d	g3b	soil wetness
`wsmx`	229	g	m	2d	g3b	field capacity of soil
`xi`	154	g	—	3d	gl	fractional cloud ice
`xivi`	150	g̅	kg/m^2	2d	g3b	vertically integrated cloud ice
`xl`	153	g	—	3d	gl	fractional cloud water
`xlvi`	231	g̅	kg/m^2	2d	g3b	vertically integrated cloud water

2.4.2 Output file forcing

The forcing output file contains the instantaneous radiative aerosol forcing if it was required by the setting of the corresponding namelist parameters. In the table of the output variables, we denote the net short wave radiation flux under clear sky conditions by $F^{\top}_{\mathrm{sw,clear}}$ at the top of any model layer and by $F^{\perp}_{\mathrm{sw,clear}}$ at the bottom of this layer. Similarly, we symbolize the net short wave radiation flux under all sky condition at the top of any model layer by $F^{\top}_{\mathrm{sw,all}}$ and by $F^{\perp}_{\mathrm{sw,all}}$ at its bottom. The corresponding quantities for thermal radiation are denoted by $F^{\top}_{\mathrm{lw,clear}}$, $F^{\perp}_{\mathrm{lw,clear}}$, $F^{\top}_{\mathrm{lw,all}}$, and $F^{\perp}_{\mathrm{lw,all}}$, respectively. A superscript 0 is added if these quantities are meant for an atmosphere free of aerosols: $F^{\top,0}_{\mathrm{sw,clear}}$, $F^{\perp,0}_{\mathrm{sw,clear}}$, $F^{\top,0}_{\mathrm{sw,all}}$, $F^{\perp,0}_{\mathrm{sw,all}}$, $F^{\top,0}_{\mathrm{lw,clear}}$, $F^{\perp,0}_{\mathrm{lw,clear}}$, $F^{\top,0}_{\mathrm{lw,all}}$, $F^{\perp,0}_{\mathrm{lw,all}}$. With a certain conversion factor c_{h}, the heating rates with and without aerosols can be obtained from the radiation fluxes. The subscript sw indicates quantities calculated for the solar radiation and lw indicates quantities calculated for the thermal radiation range:

$$
\begin{aligned}
T'_{\mathrm{sw}} &:= (F^{\top}_{\mathrm{sw,all}} - F^{\perp}_{\mathrm{sw,all}})c_{\mathrm{h}} \\
T'_{\mathrm{lw}} &:= (F^{\top}_{\mathrm{lw,all}} - F^{\perp}_{\mathrm{lw,all}})c_{\mathrm{h}} \\
T'^{0}_{\mathrm{sw}} &:= (F^{\top,0}_{\mathrm{sw,all}} - F^{\perp,0}_{\mathrm{sw,all}})c_{\mathrm{h}} \\
T'^{0}_{\mathrm{lw}} &:= (F^{\top,0}_{\mathrm{lw,all}} - F^{\perp,0}_{\mathrm{lw,all}})c_{\mathrm{h}}
\end{aligned}
$$

From these quantities, we obtain the heating rate forcing or heating rate anomalies $\Delta T'_{\mathrm{sw}}$ and $\Delta T'_{\mathrm{lw}}$ for solar and thermal radiation:

$$
\begin{aligned}
\Delta T'_{\mathrm{sw}} &:= T'_{\mathrm{sw}} - T'^{0}_{\mathrm{sw}} \\
\Delta T'_{\mathrm{lw}} &:= T'_{\mathrm{lw}} - T'^{0}_{\mathrm{lw}}
\end{aligned}
$$

Table 2.31: Output file `forcing`. The type of the output fields can be g (instantaneous grid point variable), $\overline{\mathrm{g}}$ (mean value over the output interval of grid point variable), s (spectral space variable). The dimension is either 2d (variable depends on longitudes and latitudes only), 3d (variable depends on longitudes, latitudes, and levels).

Name	Code	Type	Unit	Dimension	Stream	Explanation
`aps`				see Tab. 2.30		
`d_aflx_lw`	25	$\overline{\mathrm{g}}$	W/m^2	3d	forcing	$F^{\top}_{\mathrm{lw,all}} - F^{\top,0}_{\mathrm{lw,all}}$
`d_aflx_lwc`	26	$\overline{\mathrm{g}}$	W/m^2	3d	forcing	$F^{\top}_{\mathrm{lw,clear}} - F^{\top,0}_{\mathrm{lw,clear}}$
`d_aflx_sw`	15	$\overline{\mathrm{g}}$	W/m^2	3d	forcing	$F^{\top}_{\mathrm{sw,all}} - F^{\top,0}_{\mathrm{sw,all}}$
`d_aflx_swc`	16	$\overline{\mathrm{g}}$	W/m^2	3d	forcing	$F^{\top}_{\mathrm{sw,clear}} - F^{\top,0}_{\mathrm{sw,clear}}$
`FLW_CLEAR_SUR`	23	$\overline{\mathrm{g}}$	W/m^2	2d	forcing	$F^{\perp}_{\mathrm{lw,clear}} - F^{\perp,0}_{\mathrm{lw,clear}}$ at the surface
`FLW_CLEAR_TOP`	21	$\overline{\mathrm{g}}$	W/m^2	2d	forcing	$F^{\top}_{\mathrm{lw,clear}} - F^{\top,0}_{\mathrm{lw,clear}}$ at the top of the atmosphere
`FLW_TOTAL_SUR`	23	$\overline{\mathrm{g}}$	W/m^2	2d	forcing	$F^{\perp}_{\mathrm{lw,all}} - F^{\perp,0}_{\mathrm{lw,all}}$ at the surface

table continued on next page

Table 2.31: Output file `forcing` — continued

`FLW_TOTAL_TOP`	22	$\overline{g}$	W/m^2	2d	forcing	$F^{\top}_{\mathrm{lw,all}} - F^{\top,0}_{\mathrm{lw,all}}$ at the top of the atmosphere
`FSW_CLEAR_SUR`	13	$\overline{g}$	W/m^2	2d	forcing	$F^{\perp}_{\mathrm{sw,clear}} - F^{\perp,0}_{\mathrm{sw,clear}}$ at the surface
`FSW_CLEAR_TOP`	11	$\overline{g}$	W/m^2	2d	forcing	$F^{\top}_{\mathrm{sw,clear}} - F^{\top,0}_{\mathrm{sw,clear}}$ at the top of the atmosphere
`FSW_TOTAL_SUR`	14	$\overline{g}$	W/m^2	2d	forcing	$F^{\perp}_{\mathrm{sw,all}} - F^{\perp,0}_{\mathrm{sw,all}}$ at the surface
`FSW_TOTAL_TOP`	12	$\overline{g}$	W/m^2	2d	forcing	$F^{\top}_{\mathrm{sw,all}} - F^{\top,0}_{\mathrm{sw,all}}$ at the top of the atmosphere
`gboxarea`				see Tab. 2.30		
`geosp`				see Tab. 2.30		
`lsp`				see Tab. 2.30		
`netht_lw`	27	$\overline{g}$	K/d	3d	forcing	$\Delta T'_{\mathrm{lw}}$
`netht_sw`	17	$\overline{g}$	K/d	3d	forcing	$\Delta T'_{\mathrm{sw}}$

2.4.3 Output file tdiag

Wind, temperature, and moisture tendencies due to various processes are collected in this output file. All the tendencies are instantaneous values the mean values of which may be calculated during a model run using the mean value stream. The actual content of the tdiag output file depends on the exact choice of output variables in the `tdiagctl` namelist (see Sec. 2.2.1.17).

Table 2.32: Output file `tdiag`. The type of the output fields can be g (instantaneous grid point variable), $\overline{g}$ (mean value over the output interval of grid point variable), s (spectral space variable). The dimension is either 2d (variable depends on longitudes and latitudes only), 3d (variable depends on longitudes, latitudes, and levels).

Name	Code	Type	Unit	Dimension	Stream	Explanation
`aps`				see Tab. 2.30		
`dqdt_cloud`	36	g	K/d	3d	tdiag	dq/dt due to processes computed by the subroutine `cloud`
`dqdt_cucall`	35	g	K/d	3d	tdiag	dq/dt due to processes computed by the subroutine `cucall` (convective clouds)
`dqdt_vdiff`	31	g	K/d	3d	tdiag	dq/dt due to processes computed by the subroutine `vdiff` (vertical diffusion)
`dtdt_cloud`	6	g	K/d	3d	tdiag	dT/dt due to processes computed by the subroutine `cloud`
`dtdt_cucall`	5	g	K/d	3d	tdiag	dT/dt due to processes computed by the subroutine `cucall` (convective clouds)

table continued on next page

Table 2.32: Output file `tdiag` — continued

`dtdt_hines`	3	g	K/d	3d	tdiag	dT/dt due to processes computed by the Hines gravity wave parametrization
`dtdt_rheat_lw`	72	g	K/d	3d	tdiag	dT/dt due to radiative heating caused by radiation in the thermal spectral range
`dtdt_rheat_sw`	62	g	K/d	3d	tdiag	dT/dt due to radiative heating caused by radiation in the solar spectral range
`dtdt_sso`	4	g	K/d	3d	tdiag	dT/dt due to gravity wave drag
`dtdt_vdiff`	1	g	K/d	3d	tdiag	dT/dt due to processes computed by the subroutine `vdiff` (vertical diffusion)
`dudt_cucall`	15	g	K/d	3d	tdiag	du/dt (zonal wind component) due to processes computed by the subroutine `cucall` (convective clouds)
`dudt_hines`	13	g	K/d	3d	tdiag	du/dt (zonal wind component) due to processes computed by the Hines gravity wave parametrization
`dudt_sso`	14	g	K/d	3d	tdiag	du/dt (zonal wind component) due to gravity wave drag
`dudt_vdiff`	11	g	K/d	3d	tdiag	du/dt (zonal wind component) due to processes computed by the subroutine `vdiff` (vertical diffusion)
`dvdt_cucall`	25	g	K/d	3d	tdiag	dv/dt (meridional wind component) due to processes computed by the subroutine `cucall` (convective clouds)
`dvdt_hines`	23	g	K/d	3d	tdiag	dv/dt (zonal wind component) due to processes computed by the Hines gravity wave parametrization
`dvdt_sso`	24	g	K/d	3d	tdiag	dv/dt (zonal wind component) due to gravity wave drag
`dvdt_vdiff`	21	g	K/d	3d	tdiag	du/dt (zonal wind component) due to processes computed by the subroutine `vdiff` (vertical diffusion)

table continued on next page

Table 2.32: Output file `tdiag` — continued

`dxidt_cloud`	56	g	K/d	3d	tdiag	dx_i/dt (cloud water ice) due to processes computed by the subroutine `cloud`
`dxidt_vdiff`	51	g	K/d	3d	tdiag	dx_i/dt (cloud water ice) due to processes computed by the subroutine `vdiff` (vertical diffusion)
`dxldt_cloud`	46	g	K/d	3d	tdiag	dx_l/dt (cloud water) due to processes computed by the subroutine `cloud`
`dxldt_vdiff`	41	g	K/d	3d	tdiag	dx_l/dt (cloud water) due to processes computed by the subroutine `vdiff` (vertical diffusion)
`gboxarea`				see Tab. 2.30		
`geosp`				see Tab. 2.30		
`lsp`				see Tab. 2.30		
`st`				see Tab. 2.30		
`tm1`				see Tab. 2.30		

2.5 Run scripts

2.5.1 Systematic technical testing of ECHAM6

In many cases, scientists wish to modify the ECHAM6 code for their special applications. Before any "production" simulation can be started, the modified ECHAM6 version has to be tested thoroughly. The purpose of this collection of korn shell scripts is to provide a systematic and easy to use test bed of the ECHAM6 code on a technical level. These test scripts perform very short simulations in the T31L39 resolution over 12 time steps in different model configurations in order to trap errors in the code that cause technical malfunctions. However, this kind of tests can not detect any scientific failure or evaluate the scientific quality of the results. The tests rely on a comparison of the output of 12 time steps using the `cdo diff` tool. We apply the term that the results of two simulations are "bit identical" if the `cdo diff` command does not find differences between all netcdf or GRIB output files of these two simulations. This means that the output on the standard output device of these two simulations is allowed to be different, e.g. by new messages for a newly built in submodel facility. Futhermore, it is only checked whether the netcdf representation of the output of the two simulations is bit–identical but not whether all variables during the run of the ECHAM6 program have bit identical values in both simulations. In addition to tests on one model version that will be called the test version, such a test version of the model can be compared to a reference version in the so–called update test.

The package of scripts performing these tests can be used on various computers without queueing system and can be modified in such a way that individual namelists and input data can be provided to the test and reference model.

The following tests and combinations of them can be performed by the test tool (including checkout and compilation of the model which is always performed):

compile: This is not a real test. The respective test version is checked out from the svn version control system if necessary and compiled, but no run is performed.

single test: In this test, the test version is (checked out, compiled, and) run for 12 time steps. The test is successful if the program does not crash.

parallel test: For this test, a simulation of the test ECHAM6 version over 12 time steps is performed on 1 and 2 processors, respectively, and the result is compared by the `cdo diff` tool for every time step. The test simulation on a single processor is also performed using the parallel mode of the program. It is therefore not a test for the version of ECHAM6 without message passing interface (mpi). With this kind of test, possible parallelization errors can be detected like the usage of variables or fields which were not sent to all processors. The result of these two simulations should be bit identical. On massive parallel machines, using a lot of processors distributed over several nodes further problems may occur even if this test is passed. Such problems are often either subtle errors in the usage of mpi or compiler problems. Supplemental tests have to be performed on a later stage when the program is ported to such a platform.

nproma test: The section of the globe that is present on a processor after distribution of the data onto the processors, is vectorized by blocks of maximum length `nproma`. This means that — even if only one processor is used — surface fields of the earth do not simply have two dimensions of the size of longitudes n_{lon} and latitudes n_{lat} but are reshaped to `ngpblks` blocks of maximum length `nproma`. Since `nproma` may not be a divisor of $n_{\text{lon}} \times n_{\text{lat}}$, there may be a last block that contains fewer than `nproma` elements. This may lead to problems in the code, if such non–initialized elements of the last block are used accidentally. The nproma test traps such errors by using two different nproma lengths of 17 and 23 which are both not divisors of n_{lon} in the T31 resolution in the test simulations and comparing the results of 12 time steps. The results should be bit–identical.

rerun test: ECHAM6 has the possibility to split up a long term simulation into several runs of a shorter time period and to restart the model at a certain date. The results after restart are bit identical with those of a simulation without restart. There is a large variety of errors associated with a failure of the restart facility which can not all be trapped by this test like wrong scripting of the use of transient boundary conditions, but to pass this test is a minimum requirement. The base simulation starts at 1999-12-31, 22:00:00h, writes a restart file at 23:45:00h. It stops after a total of 12 time steps. The rerun files are used to restart the program and to complete the 12 times steps. The five time steps after restart are then compared with the simulation that was not interrupted. The results should be bit–identical.

update test: This test compares the results of two simulations with different model versions (test version versus reference version). Under certain circumstances, bit–identical results may be required in this test.

submodel off test: The above standard tests are all run in a model configuration that comprises submodels (configuration similar to the CMIP–5 simulations). In some cases, one may be interested in a configuration without any submodel. This test tries to run ECHAM6 without any submodel. If two revisions are compared, the results of this model configuration are also compared for the test and reference revision.

2.5.1.1 System requirements

The ECHAM6 test scripts can be adapted to UNIX computers without queuing system. The automatic configure procedure for the model compilation has to work and the environment has to provide the possibility to run programs using message passing interface (mpi). The initial and boundary condition data of ECHAM6 have to be directly accessible in some directory. If there is no direct access to the version control system of echam (svn), individual model versions on the computer may be used in the tests, but the path name of the location of these model versions has to follow the below described conventions.

2.5.1.2 Description of the scripts

In figure 2.1, we present the flow chart of the scripts performing the test simulations of ECHAM6 and the comparison of the results. The scripts need some additional variables that are written to files by the master script `test_echam6.sh` and read from these files by the dependent scripts. The variables can be set in the master script as described in Tab. 2.33. The corresponding files must not be modified by hand. The file `c.dat` contains the module name of the C compiler, the file `fortran.dat` contains the module name of the fortran compiler, the file `mpirun.dat` contains the absolute path and name of the command to start programs using message passing interface (mpi), the file `outfiletype.dat` contains a number associated with the type of the output files (1 for GRIB format and 2 for netcdf format).

test_echam6.sh: This script contains a definition part where all the path names and the model version for the test and reference model must be set. It is also the place at which the key word for the kind of test is defined. It calls the scripts for downloading the respective model versions from svn if they are not yet present on your computer and calls the compile and test run scripts.

compile_echam6.sh: This script downloads the respective model version from the revision administration system svn if it is not yet present on your computer and compiles the model. Compilation can be forced. Note that the compiler options depend on the settings in the input scripts of the configure procedure and may be different from revision to revision. Different compiler options may lead to numerically different results although the algorithms in the code are identical!

test_*mode*.sh: This family of scripts performs the various simulations and the comparison of the results. The *mode* is one of `single`, `parallel`, `nproma`, `rerun`, `update`, `submodeloff`, `parallelnproma`, `parallelnpromarerun`, `parallelnpromarerunsubmodeloff`, `all`.

test_echam6_run.sh: General run script for echam.

test_echam6_{test,reference}_links.sh: Script that provides the links to all input and boundary condition files needed for simulations with ECHAM6. In the standard version, the two scripts are identical but allow the user to apply different files for the reference and test model, respectively.

test_echam6_{test,reference}_namelists.sh: These scripts generate the namelists for the reference and test model separately. In the standard version, these two scripts are identical. They are useful if the introduction of a new submodel requires a namelist for the test model that is different from the namelist used for the reference model.

test_diff.sh: This script performs a comparison of all output files that are common to two test simulations. It also gives a list of outputfiles that are not common to the two test simulations. If there are no results written into an output file during the 12 time steps of the test simulations, the comparison of the files with the `cdo diff` command leads to an error message that the respective file structure is unsupported.

2.5.1.3 Usage

The scripts should be copied into a directory that is different form the original ECHAM6 directory so that you can savely change them without overwriting the original. The files `*.dat` must not be changed but contain values of "global" variables to all scripts. They are described in section 2.5.1.2. The variables that have to be modified in `test_echam6.sh` are listed in table 2.33. Note that the revision specific path of the ECHAM6 model will be automatically composed as `${REF_DIR}/${REF_BRANCH}_rev${REF_REVISION}` for the reference model and as `${TEST_DIR}/${TEST_BRANCH}_rev${TEST_REVISION}` for the test model, respectively. Inside these directories, the echam model sources are expected to be in a revision independent directory `${REF_BRANCH}` and `${TEST_BRANCH}`, respectively. The simulation results will be in directories `${REF_ODIR}/0000nrev${REF_REVISION}` and `${TEST_ODIR}/0000nrev${TEST_REVISION}` for the reference and test model, respectively. The number `n` is the number of the experiment. If in such a directory, an outputfile `*.err` exists, the test tool assumes that the simulation already exists and does not perform a new simulation. The results are not removed once a test is performed in order to avoid the repetition of the same test simulation over and over again (e.g. for the reference model). If experiments have to be repeated, the corresponding directory has to be removed by hand.
The test is then started by typing `./test_echam6.sh` in the directory of the test scripts.
The links to input and boundary condition data and the input namelists for the model revisions can be modified for the reference and the test model individually by editing the scripts `test_echam6_{reference,test}_links.sh` and `test_echam6_{reference,test}_namelists.sh`, respectively. This makes this collection of test scripts rather flexible: It may be used even for models containing extensions of ECHAM6 like ECHAM6-HAM or ECHAM6-HAMMOZ.

Table 2.33: Variables of `test_echam6.sh` that have to be modified by the user of the test scripts. The variables are listed in the order of their appearance in `test_echam6.sh`. Note that the revision specific path of the ECHAM6 model will be automatically composed as `${REF_DIR}/${REF_BRANCH}_${REF_REVISION}` for the reference model and as `${TEST_DIR}/${TEST_BRANCH}_${TEST_REVISION}` for the test model, respectively.

Variable	Explanation
`SCR_DIR`	Absolute path to diretory where test scripts are located.
`OUTFILETYPE`	File type of output files. Set to 1 for GRIB format output files and to 2 for netcdf output files. It is recommended to test ECHAM6 with both output formats.
`FORTRANCOMPILER`	If a module has to be loaded in order to use the correct fortran compiler version, give the fortran compiler module here.

table continued on next page

Table 2.33: `test_echam6.sh` — continued

CCOMPILER	If a module has to be loaded in order to use the correct C compiler version, give the C compiler module here.
MPIRUN	Absolute path and command to run a program using message passing interface (mpi).
TEST_DIR, REF_DIR	Absolute base path to directory containing model versions of test and reference model, respectively. Even if the model source code is loaded from svn, this directory has to exist.
TEST_BRANCH, REF_BRANCH	name of branch of test and reference model in the revision control system svn a revision of which has to be tested, respectively.
TEST_REVISION, REF_REVISION	revision number of test and reference model revision, respectively.
TEST_SVN, REF_SVN	URL address of test and reference model branch in svn system, respectively. Can be omitted if model source code is on local disk.
TEST_ODIR, REF_ODIR	Absolute path where test scripts can open directories for simulation results of test and reference model, respectively. This directory has to exist.
LCOMP	LCOMP=.true. forces compilation, with LCOMP=.false. compilation is done only if executable is not existing.
MODE	One of compile, single, parallel, nproma, rerun, update, submodeloff, parallelnproma, parallelnpromarerun, parallelnpromarerunsubmodeloff, all in order to perform the corresponding tests.

If some step or test was not successful, more information about the possible error is given in the protocol files that are written for each step. If the model was checked out from the svn system, there is a protocol file `checkout.log` of the checkout procedure in `${REF_DIR}/${REF_BRANCH}_${REF_REVISION}` for the reference model and `${TEST_DIR}/${TEST_BRANCH}_${TEST_REVISION}` for the test model, respectively. The configure procedure and compilation is protocolled inside the `${BRANCH}` directory of the aforementioned paths in the files `config.log` and `compile.log`, respectively. Information about each simulation can be found inside the directories `${REF_ODIR}/0000nrev${REF_REVISION}` and `${TEST_ODIR0000nrev${TEST_REVISION}` with `n` being the number of the test case indicated during the test run procedure on the screen, respectively. In these directories, the standard and standard error output of the ECHAM6 program can be found in the `0000nrev${REF_REVISION}.{log,err}` and the `0000nrev${TEST_REVISION}.{log,err}` files, respectively. The detailed result of the cdo comparison for each output file is also in these output directories in respective files `diff*.dat`. On the screen, only the most important steps and results are displayed. A certain test is successfully passed if the comparison for each file results in the message "0 of r records differ" where r is the number of records.

2.5.2 Automatic generation of runscripts for ECHAM6 on blizzard

There is a tool for the automatic generation of standard run scripts that serve to repeat some basic CMIP5 experiments in two spatial resolutions. These scripts may also serve as a starting point for more specialized experiments. These run scripts only work on `blizzard.dkrz.de` of the DKRZ computing centre and rely on certain conventions concerning directory structures and file names. A description of this tool can be found in the file `contrib/generate-scripts/README_ECHAM6` of the main echam directory.

2.5.2.1 Directory structure and file systems on `blizzard.dkrz.de`

Several file systems are accessible from the supercomputer platform `blizzard.dkrz.de` (blizzard) that all serve for different purposes. (1) There is the `$HOME` file system (located in `/pf`) that has a quota per user (8GB) and regular backups are available. This file system is good for holding the source code of the echam model and the run scripts that are used to perform a computer experiment. (2) There is a `$SCRATCH` file system (located in `/scratch`) with very fast i/o and a limited lifetime of data of 14 days currently. There is no backup available and deletion of files is automatic. This file system is good for the primary output from a model that will be treated by some postprocessing immediately after the run. It is not used by the automatically generated run scripts mentioned above. (3) There is the `/work/{PROJECT}` file system that also has fast i/o possibilities. There is no backup available, but data are not automatically deleted. There is a quota per project and NOT per user. Reasonable use of this file system requires the coordination of your work with the other members of this project. Although data are not automatically deleted, it is NOT an archive. It is meant for frequently accessed data only. (4) There are two kinds of archive systems: `/hpss/arch` and `/hpss/doku`, both accessible by `pftp`. The automatically generated run scripts make use of the following directories:
`/home/zmaw/{USER_ID}/{REPOS_NAM}`: Source code of ECHAM6. The {REPOS_NAM} is the directory `echam-6.1.00` for example.
`/home/zmaw/{USER_ID}/{REPOS_NAM}/experiments`: In this directory, a subdirectory will be created for each experiment. This subdirectory will contain a directory `scripts` in which you will find the run scripts and postprocessing scripts that were automatically generated for a particular experiment. The path contains your DKRZ user–id and a {REPOS_NAM} that can be chosen freely.
`/work/{PROJECT}/{USER_ID}/{REPOS_NAM}/experiments`: In this directory, a subdirectory will be created for the output of each experiment. Be careful to move your results into the archive as soon as you do not work with them regularly.

2.5.2.2 Generation of run scripts

Go into the directory `contrib/generate-scripts` of your ECHAM6 source code and edit the file `generate-echam.sh`. There, you only have to fill in the variables listed in Tab. 2.34.

Table 2.34: Variables needed for the automatic generation of run scripts

Variable	Explanation
`USER_ID`	user identification number of your account at DKRZ (account number)

table continued on next page

Table 2.34: `automatic scripts` — continued

`GROUP_ID`	project number of the project the work space of which you like to use for the interim storage of your simulation results
`REPOS_NAM`	The name of the directory containing the ECHAM6 source code. This directory has to be in your `$HOME` directory
`EXP_ID`	Your personal experimenter identification number. It is composed of 3 letters and a four digit number. See: http://svn.zmaw.de/dokuwiki/doku.php?id=listofids:list_of_experimenter_ids
`EXPNAME`	The experiment name determines the kind and resolution of the the experiment you are performing. Currently, only four different experiments are possible: `amip-LR` or `amip-MR` (amip experiments at either T63L47 (LR) or T63L95 (MR) spatial resolution), and `sstClim-LR` or `sstClim-MR` (experiment using climatological sea surface temperature and sea ice derived from a 500–year mean of the corresponding coupled pre–industrial control simulation at T63L47 (LR) or T63L95 (MR) spatial resolution)
`ECHAM_EXE`	Name of echam executable (normally its `echam6`)
`ACCOUNT`	Account (project) number under which computing time should be accounted (can be different of `GROUP_ID`)

2.6 Postprocessing

The ECHAM6 output is not directly suitable for visualization since some of the output fields are in the spectral space (3d–temperature, vorticity, divergence and the logarithm of the surface pressure). Furthermore, monthly or yearly mean values are more suitable for a first analysis of a simulation than instantaneous values at a certain time step. There is a standard postprocessing tool with which standard plots can be generated. This postprocessing tool also produces tables of key quantities. The postprocessing consists of two steps: (1) preparation of the ECHAM6 output data, (2) generation of the plots and tables.

2.6.0.3 Software requirements

The postprocessing scripts require the installation of the so–called "afterburner" that performs the transformation of spectral variables into grid point space and the interpolation to pressure levels, the installation of the cdo climate data operator package for mean value calculations and general manipulation of the data, the installation of the ncl NCAR graphics tool to generate the plots, and of the LaTeX program package in order to arrange the viewgraphs in one document.

2.6.0.4 Preparation of the ECHAM6 output data

The output data of an ECHAM6 simulation can be prepared for the postprocessing tool by the use of the `after.sh` script. The prerequisite is to have a simulation that was conducted over a time period of at least one complete year. The output has to be stored in monthly files. These files can contain either monthly mean values or (mean) values over smaller time intervals. It is assumed that the arithmetic mean of the output variables over the time steps in these monthly files is a good estimate of the monthly mean value. Several variables have to be modified by the user in the `after.sh` script (see Tab. 2.35).

Table 2.35: Variables of `after.sh` in alphabetical order

Variable	Explanation
`after`	Location and name of the executable of the afterburner, e.g.: `/client/bin/after`
`cdo`	Location and name of the executable of the climate data operators, e.g.: `cdo` if no search path is needed
`datdir`	Absolute path to the folder in which the original ECHAM6 simulation output files are stored
`exp`	Experiment name as defined in the variable `out_expname` of the `runctl` namelist (see Tab. 2.12)
`filename_suffix`	The extension of the monthly ECHAM6 (standard) output files after the number of the months (including leading dots), e.g.: `.01_echam.nc`. The output files can be in either GRIB format (no extension) or netcdf format (including the extension `.nc`).
`first_year`	First year of simulation data
`last_year`	Last year of simulation data
`out_format`	should be set to 1 for GRIB output format of `after.sh` (standard)
`workdir`	Absolute path to which the output files of `after.sh` are written

The output files contain monthly mean values over all simulated years as given by the `first_year` and `last_year` variable. There are 12 output files for the 3–d variables with names `ATM_${exp}_${first_year}-${last_year}_MMM` with `MMM` describing the month and 12 output files for the 2–d surface variables with names `BOT_${exp}_${first_year}-${last_year}_MMM`. These files are the input to the program that actually generates the tables and view graphs.

2.6.0.5 Generation of plots and tables

The plots and tables are generated by the script `POSTJOB` in the case of a comparison of one model simulation with era40 data or by the script `POSTJOBdiff` in the case of the comparison of two different experiments. Again, some variables have to be set by the user directly in the scripts. In the case of the script `POSTJOB` the variables are listed in Tab. 2.36, in the case of `POSTJOBdiff`, the variables are listed in Tab. 2.37.

Table 2.36: Variables of `POSTJOB` in alphabetical order

Variable	Explanation
`ATM`	= 1 if viewgraphs of atmosphere fields are desired, = 0 otherwise
`atm_RES`	Spectral resolution of the model, e.g. 31 for the T31 spectral resolution
`BOT`	= 1 if viewgraphs of surface fields are desired, = 0 otherwise
`COMMENT`	Any comment that describes your experiment (will appear on the plots)
`EXP`	Experiment name as defined in the variable `out_expname` of the `runctl` namelist (see Tab. 2.12)
`LEV`	Number of levels
`oce_RES`	Resolution of the ocean, e.g. `GR30` for the GROB 30 resolution.
`LOG`	only if `LOG_*` files exist, currently not implemented in `after.sh`

table continued on next page

Table 2.36: POSTJOB — continued

PRINTER	name of black and white printer, = 0 if printing is not desired. CAUTION: If the printer PRINTER exists, printing is automatic without asking the user again!
PRINTERC	name of color printer, = 0 if printing is not desired. CAUTION: If the printer PRINTERC exists, printing is automatic without asking the user again!
TAB	= 1 if tables are desired, = 0 otherwise
TYP	type of plots. There are 17 possible types: ANN: annual mean values (they will be calculated from the monthly means by weighting with the length of the respective months). Seasonal mean values for the seasons DJF (December, January, February), MAM (March, April, May), JJA (June, July, August), SON (September, October, November). In the case of the seasonal mean values, the length of the respective months is not taken into account when the mean values over the corresponding three months are calculated. One of the twelve months of a year (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC). The seasonal and monthly (and also annual) mean values are "climatological" mean values over possibly several years.
WORKDIR	Path to the directory where the monthly means prepared by the after.sh script are stored
YY1	First simulated year
YY2	Last simulated year

Table 2.37: Variables of POSTJOBdiff in alphabetical order for comparison of simulation 1 with simulation 2

Variable	Explanation
ATM	= 1 if viewgraphs of atmosphere fields are desired, = 0 otherwise
atm_RES	Spectral resolution of the model, e.g. 31 for the T31 spectral resolution
BOT	= 1 if viewgraphs of surface fields are desired, = 0 otherwise
COMMENT	Any comment that describes your experiment (will appear on the plots)
AEXP	Experiment name as defined in the variable out_expname of the runctl namelist (see Tab. 2.12) for simulation 1
AYY1	First simulated year of simulation 1
AYY2	Last simulated year of simulation 1
BEXP	Experiment name as defined in the variable out_expname of the runctl namelist (see Tab. 2.12) for simulation 2
BYY1	First simulated year of simulation 2
BYY2	Last simulated year of simulation 2
LEV	Number of levels
oce_RES	Resolution of the ocean, e.g. GR30 for the GROB 30 resolution.
LOG	only if LOG_* files exist, currently not implemented in after.sh
PRINTER	name of black and white printer, = 0 if printing is not desired. CAUTION: If the printer PRINTER exists, printing is automatic without asking the user again!

table continued on next page

Table 2.37: `POSTJOBdiff` — continued

`PRINTERC`	name of color printer, = 0 if printing is not desired. CAUTION: If the printer `PRINTERC` exists, printing is automatic without asking the user again!
`TAB`	= 1 if tables are desired, = 0 otherwise
`TYP`	type of plots. There are 17 possible types: `ANN`: annual mean values (they will be calculated from the monthly means by weighting with the length of the respective months). Seasonal mean values for the seasons `DJF` (December, January, February), `MAM` (March, April, May), `JJA` (June, July, August), `SON` (September, October, November). In the case of the seasonal mean values, the length of the respective months is not taken into account when the mean values over the corresponding three months are calculated. One of the twelve months of a year (`JAN`, `FEB`, `MAR`, `APR`, `MAY`, `JUN`, `JUL`, `AUG`, `SEP`, `OCT`, `NOV`, `DEC`). The seasonal and monthly (and also annual) mean values are "climatological" mean values over possibly several years.
`WORKDIR`	Path to the directory where the monthly means prepared by the `after.sh` script are stored

The results are stored in several files in the directory `${WORKDIR}_${TYP}`. The tables are in the files `tabelle_${EXP}_${YY1}-${YY2}_${TYP}[.ps]` in either ASCII or postscript (ending `.ps`) format. The viewgraphs are stored in the files `ATM_${TYP}_${EXP}.[tex,ps]`, `ATMlola_${TYP}_${EXP}.[tex,ps]`, and `BOT_${TYP}_${EXP}.[tex,ps]`. The LaTeX files `*.tex` combine several encapsulated postscript format viewgraphs in one document.

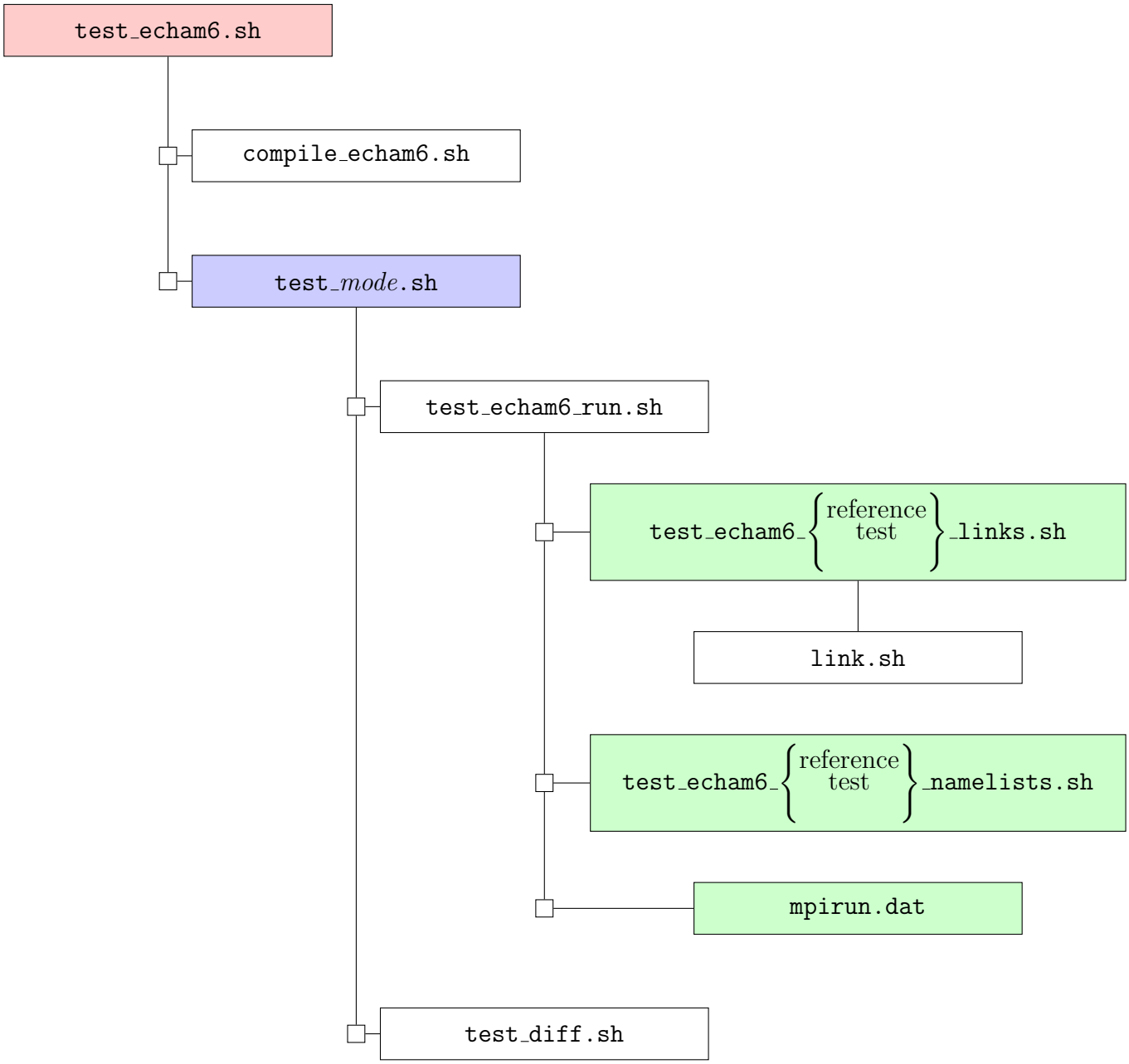

Figure 2.1: Flow chart of test scripts. The main script in the red box has to be modified by the user. The scripts in the green boxes can be modified in order to use different model settings than the standard ones for test or reference model, respectively. The script in the blue box depends on the test mode and is one of *mode*=`single`, `parallel`, `nproma`, `rerun`, `submodeloff`, `parallelnproma`, `parallelnpromarerun`, `parallelnpromarerunsubmodelloff`, `update`, `all`. The scripts with *mode*=`parallelnpromarerun`, `parallelnpromarerunsubmodeloff`, and `all` need an additional script `mve` to move the rerun files to files with new names.

Chapter 3

Technical Documentation

3.1 Parallelization

3.1.1 General description

The parallel version of ECHAM is based on a domain distribution approach, i.e. every processor only handles a limited domain of the whole globe, and only keeps the respective part of the data. In order to facilitate the distribution, the whole domain is divided into `nproca` times `nprocb` domains with `nproca` being the number of divisions in north-south direction and `nprocb` the number of divisions in east west direction. In order to achieve a good load balance in the shortwave radiation (and chemical reaction) calculations, each processor treats two parts of the globe, located opposite to each other. So half of the gridpoints of each processor will be on the daytime and the other half on the nighttime side on the globe.

Parts of the calculations within ECHAM are performed in spectral space. For these calculations the spectral coefficients are distributed over processors as well. In order to perform the Fourier and Legendre transformations - which are global operations in gridpoint and spectral space as well - two further data distributions are used, named Fourier and Legendre space. The data distributions are sketched in Figure 3.1, a more detailed discription is given in Section 3.1.3.

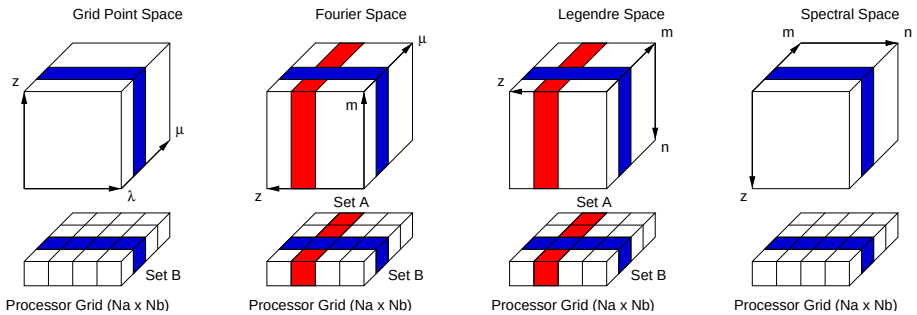

Figure 3.1: Data distribution

The data transpositions, i.e. the redistribution of data in order to perform the global Fourier and Legendre transformations are performed just before and after these transformations. All other calculations require almost no further communication (besides a few global sums) because the data required for the operations is present on the respective processor. A recipe for writing parallel routines is given in Section 3.1.2.

3.1.2 Recipe for writing or modifying parallel routines

3.1.2.1 Physical parameterizations

The physical parameterization routines (called from the routines `gpc` or `physc`) work only on one block of grid cells of consecutive longitudes. This block can be too short to accomodate all grid cells of one latitude of it may combine grid cells of more than one latitude into one block. The length of the block can be chosen arbitrarily and is called `nproma`. The loop over the blocks is performed in a higher level routine (`scan1`) and the actual block length is passed to the respective subroutines as `kproma`.
"Physics" computations at different model columns are generally independent from each other and do not require any communication between processors. Furthermore most computations do not depend on the absolute location on the globe. If these two conditions are fullfilled no further action is required for the parallel model version and a physical parameterization routine may remain unchanged. Loops over the grid cells in one block are performed by the following statement:

```
DO i=1, kproma
  ...
END DO
```

Special care must be taken if:

1. The routines are not called within the loop over blocks.

   In this case the number of longitudes and latitudes handled by the processor can be accessed by reference to the components `nglon` and `nglat` of the variable `local_decomposition` in module `mo_decompose` (cf. Section 3.1.3.2). A typical loop over blocks and block elements is given below. `dc%ngpblks` and `dc%nproma` (`dc%npromz`) are also used to specify the dimensions of local arrays.

   ```
   use mo_decomposition, only: dc => local_decomposition
   real(dp) :: xlocal (dc%nproma, dc%ngpblks) ! declare a local array
   ...
   DO j=1, dc%ngpblks-1                    ! loop over local block
     DO i=1, dc%nproma                     ! loop over grid cells in block
       ...
       xlocal (i,j) = 0._dp                ! access a local array
       ...
     END DO
   END DO
   DO i=1, dc%npromz
      ...
      xlocal (i,dc%ngpblks) = 0._dp
      ...
   END DO
   ```

2. An index to a global field is required or the absolute position on the globe must be known.

   These conditions are met in the short-wave radiation part, where the zenith angle of the sun must be calculated, or if the horizontal area covered by a column must be known, for instance in budget calculations.

Every processor handles two distinct parts of the domain on opposite sides of the globe. For this reason the first `dc%ngpblks/2` blocks are located on the northern hemisphere whereas the remaining lines are located on the southern hemisphere. The local as well as the global latitude generally runs from North to South, but some of the global arrays (for instance Gaussian weights) are still stored in so called ping-pong order (with one latitude line in the northern hemisphere being followed by the respective latitude line from the southern hemisphere).

For routines called within `gpc` or `physc` the local latitude index `jglat` and the global ping-pong index `igprow` are stored in the module variable `nrow(2)` in module `mo_control`:

```
nrow(1) = igprow ! global ping pong index
nrow(2) = jlat   ! local  index north -> south
```

3. Global sums are required.

   Global sums should be avoided, in order to prevent communication between processors. In the case that global operations cannot be avoided, routines to derive global (or zonal) sums may be found in module `mo_global_op` (cf. Section 3.1.6).

4. Dependencies between horizontal gridpoints exist.

   Dependencies between horizontal gridpoints within the physical routines should be avoided, in order to prevent communication between processors. If possible these calculations should be done at locations in the program where suitable data transpositions have already been performed or in dedicated routines (for instance in the semi–Lagrangian transport routine).

5. Input and Output

   Input and Output is addressed in Section 3.1.2.2

3.1.2.2 Input/Output

Two things must be considered when files are read or written:

1. In parallel mode, only one processor is allowed to perform I/O. This processor will also be called I/O processor. The logical variable `p_parallel_io` (from `mo_mpi`) has the value `.true.` on the I/O processor only and has the value `.false.` on all other processors. In single processor mode (indicated by a value `.false.` of `p_parallel`) the data is merely read or written.

2. The values of variables read by the I/O processor must be communicated to the other processors. If all processors are supposed to receive the same information the broadcast routine `p_bcast` (from `mo_mpi`) must be called. In case of two or three dimensional arrays each processor only holds the information relevant for its subdomain. In this case the I/O must be performed on a global variable (generally only allocated on the processor which performs I/O) different from the local variable which finally is used for computations. In order to communicate the data to processors in gridpoint space the routine `scatter_gp` from module `mo_transpose` must be called. Similar routines exist in order to distribute data in spectral space (`scatter_sp`) or do gather the data from the other processors (`gather_gp`, `gather_sp`). Generic interfaces are provided for the broadcast and gather or scatter routines (cf. Section 3.1.4) for different data types and array dimensions.

Below some examples are given. Note that generally I/O is not performed directly, but routines are provided for reading and writing specific formats (Grib, Netcdf).

1. Read and broadcast some information

   The broadcast routine requires `p_io` as actual parameter in order to identify the processor which sends the information, i.e. the processor which performs I/O.

```
USE mo_mpi, ONLY: p_parallel, p_parallel_io, p_broadcast, p_io
IF (p_parallel) THEN
   IF (p_parallel_io) THEN
      READ x
   ENDIF
   CALL p_bcast (x, p_io)
ELSE
   READ x
ENDIF
```

2. Read and scatter some information

   In this example `x` is a 3 dimensional field (`kbdim`, `levels`, `ngpblks`, where `kbdim` is the maximum length of block) which finally stores the local information on each processor. Information on the data distribution of all processors is provided in the variable `global_decomposition` and must be passed to the scatter and gather routines.

```
USE mo_mpi,       ONLY: p_parallel, p_parallel_io, p_io
USE mo_transpose, ONLY: scatter_gp
USE mo_decompose, ONLY: gl_dc => global_decomposition, &
                        dc    => local_decomposition
REAL, POINTER :: tmp (:,:,:)                ! global read buffer
REAL          :: x   (dc%nproma, dc%nlev, dc%ngpblks)
IF (p_parallel) THEN                        ! in parallel mode:
  NULLIFY(tmp)                        ! nullify global array not used
  IF(p_parallel_io) THEN
    ALLOCATE (tmp(dc%nlon,dc%nlev,dc%nlat)) ! allocate global array used
    READ x                                  ! read information
  ENDIF
  CALL scatter_gp(tmp, x, gl_dc)            ! scatter
  IF (p_parallel_io) DEALLOCATE (tmp)       ! deallocate global array
ELSE                                        ! in single processor mode:
   READ x                                   ! merely read
ENDIF
```

3. Gather and write some information

   This example is very similar to the previous one.

```
USE mo_mpi,       ONLY: p_parallel, p_parallel_io, p_io
USE mo_transpose, ONLY: gather_gp
USE mo_decompose, ONLY: gl_dc => global_decomposition, &
```

```
                          dc    => local_decomposition
REAL, POINTER :: tmp (:,:,:)                ! global read buffer
REAL          :: x   (dc% nglon, dc% nlev, dc% nglat)
IF (p_parallel) THEN                        ! in parallel mode:
  NULLIFY(tmp)                          ! nullify global array not used
  IF(p_parallel_io) THEN
    ALLOCATE (tmp(dc%nproma,dc%nlev,dc%ngpblks)) ! allocate
                                        !global array used
  ENDIF
  CALL gather_gp(tmp, x, gl_dc)             ! gather
  IF(p_parallel_io) THEN
    WRITE x                                 ! write information
    DEALLOCATE (tmp)                        ! deallocate global array
  ENDIF
ELSE                                        ! in single processor mode:
   WRITE x                                  ! merely write
ENDIF
```

3.1.3 Decomposition (mo_decompose)

The decomposition is handled by the module `mo_decompose` which is described in this section. The domain decomposition is performed by a call to the routine `decompose` with the following parameters:

`global_dc`
: Derived decomposition table (output).

`nlat, nlon, nlev`
: These parameters determine the size of the global domain: `nlat` is the number of latitudes (which must be even), `nlon` is the number of longitudes and `nlev` is the number of levels.

`nm, nn, nk`
: These parameters give the number of wavenumbers in spectral space. Currently only triangular truncation is allowed with `nm = nn = nk`.

`nproca, nprocb`
: Following the ideas of the Integrated Forecast System (IFS) of the European Centre of Midium–Range Weather Forcast (ECMWF) the total domain is covered by `nproca` times `nprocb` processors. In Gridpoint space the domain is divided into `nprocb` subdomains in east-west direction and 2 times `nproca` subdomains in north-south directions. Details are given below in the subsections of this paragraph.

  The default decomposition may be modified by the following optional parameters:

`norot`
: In order to improve load balancing in the shortwave radiation part half of the gridpoints of each processor should be exposed to the sun whereas the other half should be located at the nocturnal side of the globe. Thus each processor handles two subdomains on opposite sides of the globe. Actually the two domains must consist of latitude rows with the same absolute values of latitudes, but with opposite sign. The longitude values in the southern

domain are rotated by 180 degree with respect to the corresponding gridpoints in the northern domain. Setting this optional parameter to `.true.` the southern domain is not rotated. If the code runs on one processor this results in a continuous global domain as in the serial program version.

`lfull_m`
Setting this optional parameter to `.true.` ensures that the decomposition in spectral space does not spread wavenumbers with the same longitudinal wavenumber m over different processors. This option is not recommended because it decreases load balance in spectral space.

`debug`
Setting this optional parameter to `.true.` runs a second copy of the model using one additional processor so that `nproca` $\times$ `nprocb` $+ 1$ processors are required in this case. Furthermore it is assumed that `norot=.true.` for this additional run so that the decomposition corresponds with that of the original serial version.

The values of the variables of the two model copies are compared at certain breakpoints and further tests for equality of corresponding variables can be inserted at any time of program execution. This is the most rigorous test of the parallel version.

A value `.true.` of the logical module variable `debug_parallel` indicates that the parallel test mode is enabled.

Decomposition information is stored in the module variables `global_decomposition` and `local_decomposition` of derived type `pe_decomposed`. The elements of the array `global_decomposition` describe the decomposition for each processor. The scalar type `local_decomposition` holds the decomposition of the actual processor.
The data type `pe_decomposed` described in the subsection below holds the decomposition information for a single processor.

3.1.3.1 Information on the whole model domain

The following components of data type `pe_decomposed` have the same contents for all processors of the model:

`nlon`: number of longitudes of the global domain.

`nlat`: number of latitudes of the global domain.

`nlev`: number of levels of the global domain.

`nm`: maximum wavenumber used. Only triangular truncation is supported.

The following components depend on `nm`:

`nnp(m+1)`: number of spectral coefficients for each longitudinal wavenumber m, $m = 0, nm$

`nmp(m+1)`: displacement of the first point of m-columns within the array storing the spectral coefficients. Actually `nmp(1)=0` and `nmp(nm+2)=` last index of the array storing the spectral coefficients. The actual number of coefficiens is $2 \times nmp(nm+2)$ because 2 coefficients are stored for each wavenumber.

3.1.3.2 Information valid for all processes of a model instance

The following components of data type `pe_decomposed` have the same contents for all processors of each instance of the model:

`nprocb`: number of processors for the dimension that counts longitudes

`nproca`: number of processors for the dimension that counts latitudes

`d_nprocs`: number of processors used in the model domain $nproca \times nprocb$.

`spe, epe`: Index number of first and last processor which handles this model domain.

`mapmesh(ib,ia)`: array mapping from a logical 2-d mesh to the processor index numbers within the decomposition table `global_decomposition`. $ib = 1, nprocb$; $ia = 1, nproca$.

3.1.3.3 General Local Information

The contents of the remaining components of data type `pe_decomposed` is specific for each processor.

`pe`: processor identifier. This number is used in the `mpi` send and receive routines.

`set_b`: index of processor in the direction of logitudes. This number determines the location within the array `mapmesh`. processors with ascending numbers handle subdomains with increasing longitudes (i.e. from west to east).

`set_a`: index of processor in the direction of latitudes. This number determines the location within the array `mapmesh`. Processors with ascending numbers handle subdomains with decreasing values of absolute latitudes (i.e. from the pole to the equator within each hemisphere).

3.1.3.4 Grid space decomposition

In grid space longitudes and latitudes are spread over processors. Each processor handles all levels of a model column.

`nglat, nglon`: number of latitudes and longitudes in grid space handled by this processor.

`glats(1:2), glate(1:2)`: start and end values of global latitude indices.

`glons(1:2), glone(1:2)`: start and end values of global of longitude indices. Each processor handles two subdomains located on opposite sides of the globe. The first elements `1:nglat/2` of array dimensions indexing latitudes correspond to global latitude indices `glats(1):glate(1)`. The last elements `nglat/2+1:nglat` correspond to global latitude indices `glats(2):glate(2)`. Both, local and global latitude indices run from north to south. Elements $e(i,j), i = 1 : nglon, j = 1 : nglat/2$ of a local array correspond to elements $g(k,l), k = glons(1), glone(1), l = glats(1) : glate(1)$ of the respective global array.

`glat(1:nglat)`: global latitude index.

`glon(1:nglon)`: offset to global longitude index. These components facilitate indexing of global arrays. Elements $e(i,j), i = 1 : nglon, j = 1 : nglat/2$ of a local array correspond to elements $g(glat(i), +glon(i) + j)$ of the respective global array.

3.1.3.5 Fourier space decomposition

In order to perform the Fourier transformation, the arrays are redistributed so that each processor holds all longitudes or Fourier components. Latitudes are spread over processors as in grid space. Additionally the levels are distributed.

`nflat, nflev`: number of latitudes and levels on this processor.

`nflevp1`: number of levels plus one on this processor. If global arrays hold `nlev+1` elements per column they require `nflevp1` on this processor. `nflevp1` is equal to `nflev+1` if the last level is handled by this processor, otherwise `nflevp1` is equal to `nflev`.

`flats(2), flate(2)`: start and end values of latitudes indices. As in grid space 2 subdomains located on the northern and southern hemisphere are handled.

`flevs, fleve`: start and end values of levels. The elements $e(k), k = 1, nflevp1$ of a local array correspond to elements $g(l), l = flevs : fleve$ of the respective global array.

`lfused`: `.true.` if this processor is used in Fourier space.

3.1.3.6 Legendre space decomposition

In order to perform the Legendre transformation, the arrays are redistributed so that each processor holds all latitudes or spectral coefficients for a given longitudinal wavenumber. Levels are spread over processors as in Fourier space. Additionally the longitudinal wavenumbers are distributed.
Row of PEs with same set_a:

`nlm`: number of local longitudinal wave numbers m handled by this processor.

`lm(1:nlm)`: actual longitudinal wave numbers handled by this processor.

`lnsp`: number of complex spectral coefficients handled by this processor.

`nlmp(1:nlm)`: displacement of the first coefficient of columns (with same longitudinal wave number) within a globally indexed array (as described by components `nm`, `nnp`, `nmp`).

nlnp(1:nlm): number of points on each column with same longitudinal wave number m.

nlnm0: number of coefficients with longitudinal wave number m=0 on this processor.

Column of PEs with same set_b:

`nllev, nllevp1`: number of levels (+1) handled by this processor as in Fourier space.

`llevs, lleve`: start and end values of level indices as in Fourier space.

3.1.3.7 Spectral space decomposition

For spectral computations the arrays are redistributed so that each processor holds all levels for a given spectral coefficient. Longitudinal wavenumbers are spread over processors as in Legendre space. Remaining spectral coefficients are spread over processors.

`snsp, snsp2`: number of spectral coefficients handled by this processor and number of coefficients multiplied by 2.

`ssps, sspe`: first and last spectral coefficient with respect to the ordering in Legendre space.

`lfirstc`: true, if first global coefficient (m=0,n=0) resides on this processor.

`ifirstc`: location of first global coefficient on this processor.

`np1(1:snsp)`: value of (n+1) for all coefficients of this processor.

`mymsp(1:snsp)`: value of m for all coefficients of this processor.

`nns`: number of different n-values for this processor.

`nindex(1:nns)`: values of (n+1) different n-values for this processor.

`nsm`: number of longitudinal wavenumbers per processor.

`sm (1:nsm)`: actual longitudinal wave numbers handled by this processor.

`snnp(1:nsm)`: number of n coefficients per longitudinal wave number m.

`snn0(1:nsm)`: first coefficient n for a given m.

`nsnm0`: number of coefficients with m=0 on this processor.

3.1.4 Gather, Scatter and Low Level Transposition Routines (mo_transpose)

The module `mo_transpose` holds the routines to scatter global fields (after input) among the processors, to gather distributed fields from the processors (for output and debug purposes) and to perform the transpositions between the different decompositions (grid, Fourier, Legendre and spectral space).

3.1.4.1 Gather and Scatter routines (gather_xx, scatter_xx)

Generic interfaces are defined for specific routines to act on arrays of different rank (for 3-D atmospheric fields, 2-D surface fields, etc.). Arrays of rank 4 are supported in order to handle arrays allocated in memory buffer. The actual representation (2-D, 3-D) is derived from the shape of the rank 4 arrays or rank 3 arrays.
All scatter and gather routines have a similar interface:

```
subroutine scatter_xx (gl, lc, gl_dc)

subroutine gather_xx (gl, lc, gl_dc, [source])
```

The postfix `xx` is one of `gp`, `ls`, `sa` or `sp` and denotes the space to scatter/gather to/from.
The parameter `gl` is a pointer of rank 1 to 4 pointing to the global array. `gl` needs to be allocated only on the processor which performs i/o.
The parameter `lc` is an array of the same rank as `gl` holding the distributed array.
The parameter `gl_dc` holds the global decomposition table.
All scatter routines distribute a global array from the i/o processor to the decomposed arrays of all processors, including itself.
The gather routines have an optional parameter `source` in order to gather fields from different model copies run in parallel for debug purposes. `source` may have one of the following values:

`-1`: gather from all processors. If more than one model copy is run, the result depends on the actual I/O processor within the global decomposition table.

`0`: gather from the i/o processor only. If more than one model copy is run this is the processor which performs calculations on the whole model domain.

`1`: gather from all processors besides the I/O processor. If more than one model copy is run these processors perform the parallel calculations on the distributed domain.

`not present`: The effect is the same as if `source` had the value of the variable `debug_parallel` in `mo_decompose`.

The shape of the arrays `gl` may be one of the following:

`scatter_gp, gather_gp`: (grid space)

(nlon, nlev, ntrac, nlat)	3D tracer fields
(nlon, nlev, nlat, 1) (nlon, nlev, nlat)	3D gridpoint field
(nlon, nlat, 1, 1) (nlon, nlat, 1) (nlon, nlat)	2D surface field

`nlon`, `nlat` are the number of longitudes and latitudes of the global field `gl` as specified by the respective components of `local_decomposition`. `nlev`, `ntrac` are arbitrary numbers of vertical levels and tracers. *If more longitudes are passed only* `nlon` *or* `nglon` *longitudes are scattered/gathered.*

`scatter_sp, gather_sp`: (spectral space)

(nlev, 2, nsp, 1) (nlev, 2, nsp)	full spectral field
(nlev, nnp1, 1, 1) (nlev, nnp1, 1) (nlev, nnp1)	spectral array with m=0 coefficients only (zonal mean in grid space)

The global field `gl` has `nsp` spectral coefficients or `nnp1` coefficients for the zonal wavenumber m=0 only as specified by the respective components of `local_decomposition`. The corresponding decomposed field `lc` has `snsp` spectral coefficients or `nsnm0` coefficients for the zonal wavenumber m=0 only. `nlev` is an arbitrary number of vertical levels. The second index is 2 because 2 coefficients are stored for each wavenumber.

`scatter_sa, gather_sa`: (symmetric/assymetric Fourier components)

(nlev,	2,	nm+1,	nhgl)	full Fourier transformed field
(nlev,	nhgl,	1,	1)	Fourier transformed field (m=0 only)
(nlev,	nhgl)			(zonal mean in grid space)

For reasons of computational efficiency, Legendre transformation is performed on symmetric and asymmetric (with respect to the equator) fields seperately. The symmetric/asymmetric Fourier components are input to the Legendre transform (output of the inverse transform). Thus, the decomposition of these fields corresponds to Legendre space, i.e. vertical levels and zonal wavenumbers are spread over processors.

The global field `gl` has `nm+1` zonal wavenumbers and `nlev` or `nlev+1` vertical levels as specified by the respective components of `local_decomposition`. The corresponding decomposed field `lc` has `nlm` zonal wavenumbers and `nllev` or `nllevp1` vertical levels. `nhgl=nlat/2` is half of the number of Gaussian latitudes. The second index of the full fields is 2 because 2 coefficients are stored for each wavenumber.

`scatter_ls, gather_ls`: (Legendre space)

Scatter and gather routines to/from Legendre space are used for debugging purposes only.

(2*(nm+1),	nlev,	nlat,	nvar)	Fourier components, (gather routine only)
(nlev,	2,	nsp)		full spectral field
(nlev,	nnp1)			spectral field with m=0 only

Global Fourier transformed fields (in Legendre space distribution) have `2*(nm+1)` spectral coefficients and `nlev` or `nlev+1` vertical levels as specified by the respective components of `local_decomposition`. Global spectral fields have `nsp` spectral wavenumbers or `nnp1` coefficients for m=0 only. The corresponding decomposed field `lc` has `nlm` zonal wavenumbers or `lnsp` complex spectral coefficients and `nllev` or `nllevp1` vertical levels. `nlat` is the number of latitudes and `nvar` an arbitrary number of variables.

3.1.4.2 Transposition routines (tr_xx_yy)

The general interface of the transpose routines is:
`subroutine tr_xx_yy (gl_dc, sign, xxfields.., yyfields..)`
`TYPE (pe_decomposed) :: gl_dc` decomposition table
`INTEGER :: sign` direction of transposition: 1: xx–>yy, -1: xx<–yy
`REAL :: xxfields` fields in xx-space
`REAL :: yyfields` fields in yy-space
With xx, yy being one of `gp` (gridpoint space), `ls` (Legendre space), or `sp` (spectral space). The shape of the array arguments `xxfields`, `yyfields` depends on the data structure in the respective spaces. The specific interfaces are as follows:

```
SUBROUTINE tr_gp_fs (gl_dc, sign, gp1, gp2, gp3, gp4, gp5, gp6, gp7,&
                     sf1, sf2, sf3, zm1, zm2, zm3, fs, fs0)
!
! transpose
!   sign= 1 : grid point space  -> Fourier space
!   sign=-1 : grid point space <-  Fourier space
!
!
```

```
TYPE (pe_decomposed) ,INTENT(in)     :: gl_dc  (:)       ! decomposition
INTEGER              ,INTENT(in)     :: sign             ! 1:gp>fs; -1:gp<fs
REAL                 ,INTENT(inout)  :: gp1    (:,:,:)   ! gridpoint space 3d
                                        ...
REAL                 ,INTENT(inout)  :: gp7    (:,:,:)   !
REAL ,OPTIONAL       ,INTENT(inout)  :: sf1    (:,:)     ! gridpoint space 2d
REAL ,OPTIONAL       ,INTENT(inout)  :: sf2    (:,:)     ! gridpoint space 2d
REAL ,OPTIONAL       ,INTENT(inout)  :: sf3    (:,:)     ! gridpoint space 2d
REAL ,OPTIONAL       ,INTENT(inout)  :: zm1    (:,:)     ! zonal mean
REAL ,OPTIONAL       ,INTENT(inout)  :: zm2    (:,:)     ! zonal mean
REAL ,OPTIONAL       ,INTENT(inout)  :: zm3    (:,:)     ! zonal mean
REAL                 ,INTENT(inout)  :: fs     (:,:,:,:) ! Fourier space
REAL ,OPTIONAL       ,INTENT(inout)  :: fs0    (:,:,:)   ! zonal mean, Four.

SUBROUTINE tr_fs_ls (gl_dc, sign, fs, ls, fs0, ls0)
!
! transpose
!   sign= 1 : Fourier space  -> Legendre space
!   sign=-1 : Fourier space <-  Legendre space
!
TYPE (pe_decomposed) ,INTENT(in)     :: gl_dc  (:)       ! decomposition
INTEGER              ,INTENT(in)     :: sign             ! 1:fs>ls; -1:gs<ls
REAL                 ,INTENT(inout)  :: fs   (:,:,:,:)   ! fs
REAL                 ,INTENT(inout)  :: ls   (:,:,:,:)   ! ls
REAL ,OPTIONAL       ,INTENT(inout)  :: fs0  (:,:,:)     ! fs, zonal means
REAL ,OPTIONAL       ,INTENT(inout)  :: ls0  (:,:,:)     ! ls, zonal means

SUBROUTINE tr_ls_sp (gl_dc, sign, ls1, sp1, ls2, sp2, ls3, sp3, ls0, sp0)
!
! transpose
!   sign= 1 : Legendre space  -> spectral space
!   sign=-1 : Legendre space <-  spectral space
!
TYPE (pe_decomposed) ,INTENT(in)     :: gl_dc (:)     ! decomposition
INTEGER              ,INTENT(in)     :: sign          ! 1:ls&gtsp; -1:ls&ltsp
REAL                 ,INTENT(inout)  :: ls1   (:,:,:) ! Legendre space
REAL                 ,INTENT(inout)  :: sp1   (:,:,:) ! spectral space
                                        ...
REAL                 ,INTENT(inout)  :: ls3   (:,:,:) ! Legendre space
REAL                 ,INTENT(inout)  :: sp3   (:,:,:) ! spectral space
REAL ,OPTIONAL       ,INTENT(inout)  :: ls0   (:,:)   ! Legendre (m=0 only)
REAL ,OPTIONAL       ,INTENT(inout)  :: sp0   (:,:)   ! spectral (m=0 only)
```

3.1.5 High Level Transposition Routines (mo_call_trans)

The routines in module `mo_call_trans` gather the fields to be transposed from the respective modules and pass them as actual parameters to the routines which finally perform the transformations (defined in module `mo_transpose`). If ECHAM is run in test mode, the correctness

of the parallel implementation is tested by calling the respective routines for the ingoing and outgoing parameters. Test routines are also provided for the content of some buffers.

The fields involved in the transformation and test routines are listed below.

subroutine	spectral_to_legendre
Input :	from module mo_memory_ls (Legendre space)
ld	
ltp	
lvo	
lu0	
Output :	to module mo_memory_sp (spectral space)
sd	
stp	
svo	
su0	

subroutine	legendre_to_fourier
Input :	from module mo_buffer_fft (Legendre space)
fftl	buffer for 2D and 3D fields
lbm0	buffer for zonal means (m=0)
Output :	to module mo_buffer_fft (Fourier space)
fftz	buffer for 2D and 3D fields
fbm0	buffer for zonal means (m=0)

subroutine	fourier_to_gridpoint
Input :	from module mo_buffer_fft (Fourier space)
fftz	buffer for 2D and 3D fields
fbm0	buffer for zonal means (m=0)
Output :	to module mo_scan_buffer (gridpoint space)
d_scb	
t_scb	
u_scb	
v_scb	
vo_scb	
dtm_scb	
dtl_scb	
alps_scb	
dalpsl_scb	
dalpsm_scb	
u0_scb	
du0_scb	
ul_scb	

subroutine	gridpoint_to_fourier
Input :	from module mo_scan_buffer (gridpoint space)
rh_scb	
dm_scb	
vom_scb	
vol_scb	
u0_scb	
du0_scb	
ul_scb	
Input :	from module mo_memory_g1a (gridpoint space)
alpsm1	
dm1	
tm1	
vom1	
Output :	to module mo_buffer_fft (Fourier space)
fftz	buffer for 2D and 3D fields
fbm0	buffer for zonal means (m=0)

subroutine	fourier_to_legendre
Input :	from module mo_buffer_fft (Fourier space)
fftz	buffer for 2D and 3D fields
fbm0	buffer for zonal means (m=0)
Output :	to module mo_buffer_fft (Legendre space)
fftl	buffer for 2D and 3D fields
lbm0	buffer for zonal means (m=0)

subroutine	legendre_to_spectral
Input :	from module mo_memory_ls (Legendre space)
ld	
ltp	
lvo	
lu0	
Output :	to module mo_memory_sp (spectral space)
sd	
stp	
svo	
su0	

subroutine	test_memory_f (text)
Test :	module mo_memory_f
f	

subroutine	test_memory_gp (text)

subroutine	test_scan_buffer (gp, text)

subroutine	test_row_buffer (j, text)

3.1.6 Global operations (`mo_global_op`)

In this module, subprograms are collected that perform global operations on 2–d and 3–d fields like the calculation of global or zonal mean values. Any global operation needs communication

between the processors. Even if integrals are split into integrals over the domain that is present on each processor and the summation over all processors, the global operation subroutines slow down the ECHAM6 program the more the more processors are used in a simulation. For this performance reason, it is highly recommended to reduce global operations to a strict minimum in ECHAM6 and to perform such operations in the postprocessing step that can be performed in parallel to a longer simulation.

3.2 Data structures and memory use

3.2.1 Output Streams and Memory Buffer

3.2.1.1 Functionality

The **Output Stream** interface maintains a list of output streams. Generally one ore more streams are associated to an output file. Each stream has attributes specifying the file name, file type, etc.. It further holds a linked list of **Memory Buffer elements**, of 2 to 4 dimensional arrays and associated meta information.

3.2.1.2 Usage

First, a new output stream must be created by calling subroutine new_stream. Afterwards fields may be allocated by calling `add_stream_element`.

Create a new output stream

The access to the output stream interface is provided by module `mo_memory_base`:

```
USE mo_memory_base, ONLY: t_stream,                                  &
                          new_stream, delete_stream,                 &
                          default_stream_setting, add_stream_element, &
                          get_stream_element, set_stream_element_info, &
                          memory_info,                               &
                          ABOVESUR2, ...
```

To create a new output stream the routine `new_stream` has to be called:

```
TYPE (t_stream) ,pointer :: mystream
...
CALL new_stream (mystream ,'mystream')
```

`mystream` is a pointer holding a reference to the output stream returned by subroutine `new_stream`. `'mystream'` is the identification name of the output stream.
By default, the output and rerun filenames are derived from the name of the output stream (here 'mystream') by appending a respective suffix (here `'_mystream'`) to the standard filenames. The content of the output stream is written to the rerun file and to the output file. To change the defaults, optional parameters may be provided (cf. section 3.2.1.3).

Add a field to an output stream

To include items in the output stream `mystream` the routine `add_stream_element` has to be called. A unique name must be given to identify the quantity and a pointer associated to the field is returned. For example, to add a surface field `a` and an atmospheric field `b` with names `'A'` and `'B'`, the following sequence of subroutine calls is required:

```
REAL, POINTER :: a (:,:)
REAL, POINTER :: b (:,:,:)
REAL, POINTER :: c (:,:)
...
CALL add_stream_element (mystream, 'A' ,a )
CALL add_stream_element (mystream, 'B' ,b )
```

By default suitable sizes are assumed for surface (2-d pointer `a`) or atmospheric fields (3-d pointer `b`). To choose other sizes (e.g. spectral fields or a non-standard number of vertical layers) optional parameters must be specified. The specification of the optional parameters is given in section 3.2.1.4

A routine is available to associate a pointer (here `c`) with an item (here `'A'`) already included in the list (previously by another sub-model for example):

```
CALL get_stream_element (mystream, 'A', c)
```

If stream element `'A'` has not been created beforehand, a null pointer is returned for `c`.

3.2.1.3 Create an output stream

Optional parameters may be passed to subroutines `new_stream` and `add_stream_element` in order to specify the attributes of output streams and memory buffers. Furthermore, routines are available to change default values for optional parameters.

The interface of the routine to create an output stream is:

SUBROUTINE new_stream			(stream ,name [,filetype] [,post_suf] [,rest_suf] [,init_suf] [,lpost] [,lpout] [,lrerun] [,lcontnorest] [,linit] [,interval])	
name	type	intent	default	description
stream	type(t_stream)	pointer		Returned reference to the new output stream.
name	character(len=*)	in		Name of the new output stream.
[filetype]	integer	in	out_filetype	Type of output file. The default (GRIB) may be changed in namelist /SDSCTL/. Alternatively NETCDF may be passed.
[post_suf]	character(len=*)	in	'_'//name	Suffix of the output file associated with the stream. The default is derived from the name of the output stream.
[rest_suf]	character(len=*)	in	'_'//name	Suffix of the rerun file.
[init_suf]	character(len=*)	in	'_'//name	Suffix of initial file.
[lpost]	logical	in	.true.	Postprocessing flag. If .true. an output file is created for this stream.
[lpout]	logical	in	.true.	Output flag. The stream is written to the output file if `lpout=.true`
[lrerun]	logical	in	.true.	If .true. the stream is read/written from/to the rerun file.
[lcontnorest]	logical	in	—	Continue a restart even if this stream is not present in any rerun file.
[linit]	logical	in	.true.	Write to initial file (does not work?)
[interval]	type(io_time_event)	in	putdata	Postprocessing output interval. Default: 12 hours.

Optional parameters are given in brackets [ ]. They should always be passed by keyword because the number and ordering of optional parameters may change.

Valid values for the argument `out_filetype` are defined within module `mo_memory_base`:

```
INTEGER ,PARAMETER :: GRIB      = 1
INTEGER ,PARAMETER :: NETCDF    = 2
```

For specification of a non-standard output time interval data type `io_time_event` (defined in module `mo_time_event`) has to be passed as argument `interval`. For example, in order to write every time step or in 6 hourly intervals, specify: `interval=io_time_event(1,'steps','first',0)` or `(6,'hours','first',0)`, respectively.

Once a stream has been created, a reference can be obtained by calling subroutine **get_stream**:

SUBROUTINE get_stream (stream ,name)				
name	type	intent	default	description
stream	type(t_stream)	pointer		Returned reference to the output stream.
name	character(len=*)	in		Name of the output stream.

3.2.1.4 Add a field to the output stream

The routine to add new elements to the output stream is:

SUBROUTINE add_stream_element			(stream ,name ,ptr [,ldims] [,gdims] [,klev] [,ktrac] [,units] [,longname] [,repr] [,lpost] [,laccu] [lmiss,] [missval,] [,reset] [,lrerun] [,contnorest] [,table] [,code] [,bits] [,leveltype] [,dimnames] [,mem_info] [,p4] [,no_default] [,verbose])	
name	type	intent	default	description
mandatory arguments :				
stream	type(t_stream)	inout		Output stream.
name	character(len=*)	in		Name of the field to add to the output stream.
ptr	real(:,:[,:][,:])	pointer		Returned reference to the memory of the 2- or 3- or 4-dimensional field.
specification of dimensions :				
[ldims(:)]	integer	in	cf. text	Local size on actual processor.
[gdims(:)]	integer	in	cf. text	Global size of the field.
[klev]	integer	in	cf. text	Number of vertical levels.
[ktrac]	integer	in	0	Number of tracers.
[repr]	integer	in	GRIDPOINT	Representation.
[leveltype]	integer	in	cf. text	Dimension index of the vertical coordinate.
postprocessing flags :				
[lpost]	logical	in	.false.	Write the field to the postprocessing file.
[laccu]	logical	in	.false.	"Accumulation" flag: Does no accumulation but divides variable by the number of seconds of the output interval and resets it to 0 after output.
[reset]	real	in	0.	Reset field to this value after output (default is zero).
rerun flags :				
[lrerun]	logical	in	.false.	Flag to read/write field from/to the rerun file.
[contnorest]	logical	in	.false.	If `contnorest=.true.`, continue restart, stop otherwise.
attributes for NetCDF output :				
[units]	character(len=*)	in	' '	Physical units.
[longname]	character(len=*)	in	' '	Long name.
[dimnames(:)]	character(len=*)	in	'lon'['lev'],'lat'	Dimension names.
attributes for GRIB output :				
[table]	integer	in	0	table number.
[code]	integer	in	0	code number.
[bits]	integer	in	16	number of bits used for encoding.
Missing values :				
[lmiss]	logical	in	.false.	If `lmiss=.true.`, missing values are set to `missval`, not set at all otherwise.
[missval]	real	in	-9×10^{33}	missing value.
miscellaneous arguments :				
[mem_info]	type(memory_info)	pointer		Reference to meta data information.
[p4(:,:,:,:)]	real	pointer		Pointer to allocated memory provided.
[no_default]	logical	in	.false.	Default values usage flag.
[verbose]	logical	in	.false.	Produce diagnostic printout.

Most arguments of the routine are optional. They may be given for the following purposes:

specification of dimensions:
The total size of the field is specified by the parameter `gdims`. In a parallel environment, the part allocated on a processor element is specified by the parameter `ldims`. The order of dimensions is (lon,lat) for 2–d, (lon,lev,lat) for 3–d and (lon,lev,any,lat) for 4–dimensional gridpoint fields. The number of size of `gdims` and `ldims` corresponds to the rank of `ptr(:,:)`.

Generally, it is not necessary to give dimension information. The sizes of the fields are derived from the model field sizes. If a 2–dimensional pointer `ptr(:,:)` is provided for `ptr`, a SURFACE field is assumed. If a 3-dimensional pointer `ptr(:,:,:)` is provided, a HYBRID field (lon,lev,lat) is assumed.

For the following cases optional arguments must be specified to overwrite the defaults:

The number of vertical levels differs from the number of model levels

To specify a number of levels different from the standard σ–hybrid co–ordinate system used in the model, the parameter `klev` may be specified. A HYBRID coordinate system is assumed in this case. However if the field is written to the postprocessing file (`lpost=.true.`), it is recommended to either pass a dimension index to parameter `leveltype` or the name of the dimensions to `dimnames` in order to pass proper attributes to the NetCDF and GRIB writing routines.

For the usual cases, dimension indices are predefined (cf. table 3.1) and may be accessed from module `mo_netcdf`. New dimensions may be defined by the use of the subroutine `add_dim` as described in section 3.2.1.8.

The field is not a gridpoint field
For non Gaussian gridpoint fields appropriate values should be passed as parameter `repr`. Predefined values (`mo_linked_list`) are:

```
INTEGER ,PARAMETER :: UNKNOWN   = -huge(0)
INTEGER ,PARAMETER :: GAUSSIAN  = 1
INTEGER ,PARAMETER :: FOURIER   = 2
INTEGER ,PARAMETER :: SPECTRAL  = 3
INTEGER ,PARAMETER :: HEXAGONAL = 4
INTEGER ,PARAMETER :: LAND      = 5
INTEGER ,PARAMETER :: GRIDPOINT = GAUSSIAN
```

In all other cases, `gdims` and `ldims` have to be defined explicitly.

postprocessing flags:
In order to write a field to an output file, `lpost=.true.` must be specified. Generally the actual values of the field are written. However, if `laccu=.true.` is specified, the values are divided by the number of seconds of the output interval before output and set to the value of the variable `reset` afterwards. The default is 0. In this case the fields should be incremented at each time step with values multiplied by the time step length in order to write temporarily averaged values to the output file. If the field is set to the maximum or minimum value during the output time period, values of `reset=-huge(0.)` or `reset=huge(0.)` shall be passed.

rerun flags:
To include the field in the rerun files, `lrerun=.true.` must be specified.

attributes for NetCDF output:
For NetCDF output, the physical units, long name, and dimension names of the field should be provided.

attributes for GRIB output:
For GRIB output, a table number and code number is required. Appropriate table and code numbers are proposed in section **??**. A predefined value `AUTO` may be passed as parameter `code` in order to automatically generate unique GRIB code numbers. The number of bits used for encoding may be changed by argument `bits`.

miscellaneous arguments:
If `verbose=.true.` is specified, a printout is generated.

The default values of the optional parameters may be changed by calling the subroutine `default_stream_setting` as described below. However if `no_defaults=.true.` is specified, these changed default values will not be used.

Generally memory is allocated for the argument `ptr` when calling `add_stream_element`, but memory may be provided externally by passing it via the argument `p4`. Even if 2–dimensional or 3–dimensional arrays are accessed via `ptr`, 4–dimensional fields are used internally and must be passed for `p4` (with dimension sizes (lon,lat,1,1) or (lon,lev,lat,1), respectively).

Meta data information about memory may be accessed by the argument `mem_info`.

3.2.1.5 Change of default values for optional arguments

The default values for the optional arguments of subroutine `add_stream_entry` may be changed for all subsequent calls related to an output stream by calling the subroutine `default_stream_setting`. This subroutine accepts the same arguments as subroutine `add_stream_entry`:

```
SUBROUTINE default_stream_setting  (stream [,units] [,ldims] [,gdims] [,repr]
                                   [,lpost] [,laccu] [,reset] [,lrerun]
                                   [,contnorest] [,table] [,code] [,bits]
                                   [,leveltype] [,dimnames] [,no_default])
```

If `no_default=.true.` is not given, previously changed default values are kept.
Properties and attributes of an existing stream element may be changed by calling `set_stream_element_info`. Again, the arguments are similar to those of `add_stream_element_info`:

```
set_stream_element_info  (stream ,name ,longname [,units] [,ldims]
                         [,gdims] [,ndim] [,klev] [,ktrac] [,alloc]
                         [,repr] [,lpost] [,laccu] [,lmiss]
                         [,missval] [,reset] [,lrerun] [,contnorest]
                         [,table] [,code] [,bits] [,leveltype]
                         [,dimnames] [,no_default])
```

3.2.1.6 Access to stream elements

References to previously defined stream elements or to their meta data can be obtained by calling the subroutine `get_stream_element` or `get_stream_element_info`, respectively:

get_stream_element_info	(stream, name, info)		
name	type	intent	description
stream	type(t_stream)	in	output stream to which reference has to be added.
name	character(len=*)	in	name of stream element.
info	type(memory_info)	out	copy of meta data type content.

get_stream_element	(stream, name, ptr)		
name	type	intent	description
stream	type(t_stream)	in	output stream list.
name	character(len=*)	in	name of stream element.
ptr	real(:,:[,:][,:])	pointer	returned reference to stream element memory.

3.2.1.7 Doubling of stream element entries

It is possible to add a reference to an output stream element to another output stream. By calling the subroutine `add_stream_reference`. This is useful when the same field shall be written to different output files.

add_stream_reference		(stream ,name [,fromstream] [,lpost] [,kprec)]	
name	type	intent	description
stream	type(t_stream)	inout	output stream list to extend.
name	character(len=*)	in	name of stream element to add.
[fromstream]	character(len=*)	in	name of output stream to take the element from.
[lpost]	logical	in	postprocessing flag of the output stream reference.
[kprec]	integer	in	precision of GRIB format in bits (default: 16).

3.2.1.8 Definition of new dimensions

If other dimensions are required than those defined in Table 3.1, new dimensions can be defined by calling the subroutine `add_dim` defined in module `mo_netcdf`.

SUBROUTINE add_dim		(name ,len [,longname] [,units] [,levtyp] [,single] [,value] [,indx])		
name	type	intent	default	description
name	character(len=*)	in		name of dimension.
len	integer	in		size of dimension.
[longname]	character(len=*)	in	' '	long name of dimension.
[units]	character(len=*)	in	' '	physical units of dimension.
[levtyp]	integer	in	0	GRIB level type.
[single]	logical	in	.false.	flag indicating single level fields.
[value]	real	in	1,2,...	values of dimension field.
[indx]	integer	out		index to be passed as argument `leveltype` to subroutine `add_stream_element`.

dimension index	name	klev	GRIB leveltype	values	units	longname
HYBRID	"lev"	nlev	109	1,...,nlev		hybrid level at layer midpoints
HYBRID_H	"ilev"	nlev+1	109	1,...,nlev+1		hybrid level at layer interfaces
SURFACE	"surface"	1	1	0		surface field
ABOVESUR2	"2m"	1	105	0	m	level 2m above the surface
ABOVESUR10	"10m"	1	105	0	m	level 10m above the surface
BELOWSUR	"jpgrnd"	5	111	3,19,78,268,698	cm	levels below the surface
TILES	"tiles"	ntiles	70	1,...,ntiles		land surface tile
SOILLEV	"soil_layer"	nsoil	71	1	cm	soil levels (water)
ROOTZONE	"root_zones"	nroot_zones	72	1,...,nroot_zones		root zone
CANOPY	"canopy_layer"	ncanopy	73	1,...,ncanopy		layers in canopy

Table 3.1: Predefined dimensions

3.3 Date and time variables

In a general atmospheric circulation model such as ECHAM6 that can be used for simulations of historic time periods but also in a "climate mode" for prehistorical time periods together with an ocean model, the orbit of the Earth around the sun has to be rather flexible. The solar irradiance is closely linked to the orbit. From the perspective of the Earth, certain aspects of the orbit can be described with the help of a calendar. There are two different orbits implemented in ECHAM6: An orbit with strictly 360 days of 24 hours in a year and another orbit that can be characterized as proleptic Gregorian meaning that the Gregorian calendar of our days is applied back to the past. Consequently, the historic dates before the 15th October 1582 are different from those of the proleptic Gregorian calendar. E.g., historically, there is no 14th October 1582, but this date is identified with the 4th October 1582 of the historic Julian calendar. The proleptic Gregorian calendar goes back to 4712/01/01 12:00:00 UTC time B.C. including a year 0. Fortran90 data structures are ideal to store and manipulate the heterogeneous structure of time expressed in a calendar date and time of a day. We describe these data structures and their usage in the following

3.3.1 Date–time variables in ECHAM6

The date and time of the Gregorian proleptic calendar can be represented in various ways leading to the following definitions of date–time (DT) variables: `time_days`, `time_intern`, `time_native`. Their definition can be found in `mo_time_conversion.f90`.

Listing 3.1: time_days

```
type time_days
! ...
  integer :: day     ! day in the proleptic Gregorian
                     ! calendar since 4712/01/01 B.C.
  integer :: second  ! second in the day [0, 86399]
end type time_days
```

Listing 3.2: time_intern

```
type time_intern
! ...
  integer :: ymd     ! ‘year month day‘ of the proleptic
                     ! Gregorian calendar
                     ! (leading zeros omitted);
                     ! e.g. 2001008 is the 8th of Oct. 200.
 integer :: hms      ! ‘hour minute second‘ of ymd
                     ! (leading zeros omitted);
end type time_intern
```

Listing 3.3: time_native

```
type time_native
! ...
  integer :: year, month, day, hour, minute, second
end type_native
```

One can also use an array of 6 elements containing year, month, hour, minute, second.

For the composed data types `time_days`, `time_intern`, and `time_native`, a direct access of the components is not possible because they are declared being “PRIVATE”. Instead, they are accessible by the use of subprograms defined in `mo_time_conversion.f90`. The reason for this is the fact that it is easy to create dates and times that is not valid. Then, all subroutines using such an invalid DT–variable would fail. In order to avoid this, all the subroutines changing one of the components of the DT–variables test whether the resulting dates and times are correct.

3.3.2 Usage of DT–variables

A family of overloaded subroutines and functions is provided in the module `mo_time_conversion.f90` by ECHAM6 to handle date–time variables:

- Set a DT–variable of type `time_days`, `time_native` or `time_intern` by the use of the overloaded routine `tc_set`. Example:

Listing 3.4: tc_set

```
  type(time_native) :: my_date
  call tc_set(kyear, kmonth, kday, khour, kminute, ksecond,
     mydate)
```

This call of `tc_set` will search for the special routine `set_native` that actually sets a variable of type `time_native` from the input variables `kyear`, `kmonth`, `kday`, `khour`, `kminute`, and `ksecond`.

- Conversion of a variable of one time format into another:

  There are $3 * 2 = 6$ possible conversions which can all be performed by a call of `tc_convert(var1,var2)`, `var1`, `var2` being of one of the 3 types.

- Getting components of a DT–variable

  The components of a DT–variable can be retrieved by a call to the subroutine `tc_get`. The first argument of `tc_get` is a variable of one of the DT–variable types, the following arguments are all optional. Their names are the names of the components of the corresponding DT–variable of the first argument. Example:

Listing 3.5: tc_get

```
type(time_native) :: my_date
call tc_get(my_date,year=kyear)
call tc_get(my_date,year=kyear,second=ksecond)
```

  In that case, the first call of `tc_get` only retrieves the value of the year, whereas the second call retrieves the year and the second of `my_date`.

- Comparison of DT–variables

  DT–variables can be compared using certain operators in order to know whether a certain date is before or after a second date. Fortran90 provides the possibility to overload intrinsic Fortran90 functions such as "<", ">" or "==". You can then use these operator symbols also for the comparison of user defined data types. In that case, the user has to provide an order on the domain of these variables.

Listing 3.6: overloaded operators

```
USE mo_time_conversion, ONLY: operator(<),operator(==),
   operator(>)
TYPE(time_native)          :: var1, var2
! ...
IF (var1 < var2) THEN
!...
```

The argument of the if statement is true if the date of `var1` is before the date of `var2`.

3.3.3 Information about actual date and time in ECHAM6.

There are three variables in which the time and date of the previous ($t - \Delta t$), the current (t), and the next time step ($t + \Delta t$) are stored. These variables are defined in `mo_time_control`:

Listing 3.7: date and time variables

```
type(time_days) :: previous_date, current_date, next_date
```

3.3.4 Variables describing repeated events.

The variable types of DT variables described so far are used for a representation of absolute date and time in ECHAM6. In this paragraph, the data structure associated with repeated events is presented. This data structure is used in the namelists (section 2.2) to determine the frquency of certain events. Each variable describing repeated events consist of an integer number and the unit, describing the frequency of the event. In addition, some keywords can be set which determine the position of the repeated events relative to the absolute time axis. The underlying data structure is defined in `mo_time_event`:

Listing 3.8: io_time_event

```
type io_time_event
  integer            :: counter    ! interval
  character(len=20)  :: unit       ! unit
  character(len=20)  :: adjustment ! adjustment
  integer            :: offset     ! offset
end type io_time_event
```

With the help of this data structure, we may define a variable `outfrq` that will describe the output frequency of a stream for example.

Listing 3.9: outfrq

```
type(io_time_event) :: outfrq
```

A variable of such a type can be read from the namelist like all the other variables describing repeated events (`putdata`, `putrerun`) but we also may whish to communicate it to all processors. For this purpose, there is a special subroutine `p_bcast_event` defined in `mo_time_control.f90` which is used in the following way:

Listing 3.10: p_bcast_event

```
USE mo_time_control,       ONLY: p_bcast_event
call p_bcast_event(outfrq, pe_io)
```

The call of `p_bcast_event` sends this variable to all processors. Then, the variable `outfrq` can be used in the definition of a new stream.

3.4 Submodel interface

3.4.1 Introduction

ECHAM6 allows the implementation of so–called submodels. A submodel can describe any additional physical processes that will either be linked in a one–way coupling to echam or a two–way coupling. A one–way coupling in this context means that the additional physical processes are such that they need input from the ECHAM6 base model but do not change the general circulation. One could also say that the results of such a model are derived from the ECHAM6 base model in a “diagnostic” way. If the base model is linked by a two–way coupling to a submodel, the submodel interacts with ECHAM6 and modifies the general circulation. An example for the one–way coupling would be diagnostic chemistry implemented in such a way that the chemical species are transported by the winds given by ECHAM6 and the chemical reactions are driven by the pressure, temperature, humidity and radiation simulated by ECHAM6. Nevertheless, the concentration of the chemical species would not be allowed to influence these quantities. A

two–way coupling would be introduced if the concentration of the chemical species influences the radiation by absorption of radiation for example.
The implementation of such submodels needs an interface to the submodel that provides a certain set of variables to the submodel routines. In fact, the submodel interface is a collection of dummy subroutines in ECHAM6 inside which the special subroutines of a submodel can be called. These special subroutines will not be a part of ECHAM6 but will perform all submodel specific tasks as the solution of the chemical kinetic equations for example. In addition to this submodel interface, many submodels need the introduction of tracers that are transported with the air flow like water waper is transported. These tracers are often associated with certain chemical species having specific physico–chemical properties. In general, it may occur that a certain species is represented by several tracers (e.g. various CO tracers depending on the region of emission of CO, so–called "tagged" tracers) so that every tracer has the same physico–chemical properties. Conceptually, it is better to separate the tracer properties from a list of physico–chemical species properties so that this information is present only once in the program. This avoids inconsistent definition of species properties and is therefore more user friendly. This separation is not yet finished in the current ECHAM6 version and the species data structure will therefore not be described here although it is present. As soon as this species concept has settled, this description will be added.

3.4.2 Submodel Interface

The submodel interface consists of the subroutines listed in Tab. 3.2 that are all collected in module `mo_submodel_interface.f90`.

Table 3.2: Submodel interface subroutines. The subroutines are listed in the same order as they are called in ECHAM6.

Subroutine	Called in	Explanation
`init_subm`	`initialize.f90`	Initialization of submodel. This comprises reading of specific submodel data. However, this is not the right place to read gridded fields.
`init_subm_memory`	`init_memory` of `mo_memory_streams.f90`	Allocation of memory for submodel either in streams or 2– and 3–dimensional fields.
`stepon_subm`	`stepon.f90`	Called at the beginning of a new time step. Good for reading data at regular time intervals.
`physc_subm_1`	`physc.f90`	Call in the "physics" part of calculation. The "physics" processes are processes in one column over a grid cell. This subroutine is called before the radiation calculation.
`radiation_subm_1`	`rrtm_interface` of `mo_radiation.f90`	Submodels can modify the optical properties of the atmosphere here. It is called before the radiation fluxes are calculated.

table continued on next page

Table 3.2: `Submodel interface` — continued

`radiation_subm_2`	`rrtm_interface` of `mo_radiation.f90`	Good for radiation diagnostics performed by submodels.
`vdiff_subm`	`vdiff.f90`	In this subroutine, net surface fluxes can be calculated that will be used as boundary conditions in the vertical diffusion equation. Good for surface emission fluxes and dry deposition fluxes.
`rad_heat_subm`	`radheat.f90`	Diagnostic of heating rates.
`physc_subm_2`	`physc.f90`	First interface that is good for calculation of physical processes of submodels like chemical kinetics or aerosol physics. It is called before cloud physics but after `vdiff` and `radheat`
`cuflx_subm`	`cuflux.f90`	Submodels can interfere with convection here. E.g. wet deposition of convective clouds has to be implemented here.
`cloud_subm`	`cloud.f90`	Implement interaction between cloud physics and submodels here. E.g. "wet chemistry" should be implemented here. Wet deposition of large scale precipitation has to be implemented here.
`physc_subm_3`	`physc.f90`	Second interface that is good for calculation of physical processes of submodels like chemical kinetics or aerosol physics. It is called after cloud physics.
`physc_subm_4`	`physc.f90`	This is the right place for submodel diagnostics after all physics processes are calculated.
`free_subm_memory`	`free_memory` of `mo_memory_streams.f90`	Deallocation of allocated submodel memory here is mandatory, otherwise the internal rerun process will fail. In addition, it is very important to set back all submodel switches to their default values. In particular switches that indicate that certain fields are allocated or certain data are read.

Inside these interface routines, the submodel specific routines should be called. These calls have to be implemented all into `mo_submodel_interface.f90` and the calls have to be effective if and only if the respective submodel is switched on. Since `mo_submodel_interface.f90` is part of the ECHAM6 code but the submodel routines are not, the calls should be switched off/on by compiler directives. In that case, the calls can be included in the standard version of `mo_submodel_interface.f90`. Neither an extra version of this module has to be kept by the submodel users nor any update has to be done "by hand".

The parameter lists of the submodel interface routines are described in the following subsections.

3.4.2.1 Interface of `init_subm`

Listing 3.11: init_subm

```
SUBROUTINE init_subm
```

This subroutine has no parameter list.

3.4.2.2 Interface of `init_subm_memory`

Listing 3.12: init_subm_memory

```
SUBROUTINE init_subm_memory
```

This subroutine has no parameter list. In general, the fields allocated here belong to the submodel. Since the submodel is supposed to be organized in modules, global submodel fields should be defined as module variables and can be brought to any submodel subroutine by use statements. Streams are easily accessible by their names. Nevertheless, subroutines of the kind `get_stream` or `get_stream_element` are slow and should not be used repeatedly. Instead, pointers to the stream elements can be stored as global submodel variables and used later in the program.

3.4.2.3 Interface of `stepon_subm`

Listing 3.13: stepon_subm

```
SUBROUTINE stepon_subm (current_date, next_date)
  TYPE(time_days)     :: current_date
  TYPE(time_days)     :: next_date
```

Table 3.3: Parameter list of arguments passed to `stepon_subm`

name	type	intent	description
`current_date`	`time_days`		time and date of current time step
`next_date`	`time_days`		time and date of prognostic time step

3.4.2.4 Interface of `physc_subm_1`

Listing 3.14: physc_subm_1

```
SUBROUTINE physc_subm_1 (kproma, kbdim, klev, &
                         klevp1, ktrac, krow, &
                         papm1,  paphm1,      &
                         ptm1,   ptte,        &
                         pxtm1,  pxtte,       &
                         pqm1,   pqte          )
  INTEGER,  INTENT(in)    :: kproma
  INTEGER,  INTENT(in)    :: kbdim
  INTEGER,  INTENT(in)    :: klev
```

```
INTEGER,  INTENT(in)    :: klevp1
INTEGER,  INTENT(in)    :: ktrac
INTEGER,  INTENT(in)    :: krow
REAL(dp), INTENT(in)    :: papm1 (kbdim,klev)
REAL(dp), INTENT(in)    :: paphm1(kbdim,klevp1)
REAL(dp), INTENT(in)    :: ptm1  (kbdim,klev)
REAL(dp), INTENT(in)    :: ptte  (kbdim,klev)
REAL(dp), INTENT(inout) :: pxtm1 (kbdim,klev,ktrac)
REAL(dp), INTENT(inout) :: pxtte (kbdim,klev,ktrac)
REAL(dp), INTENT(in)    :: pqm1  (kbdim,klev)
REAL(dp), INTENT(in)    :: pqte  (kbdim,klev)
```

Table 3.4: Parameter list of arguments passed to `physc_subm_1`

name	type	intent	description
`kproma`	integer	in	actual length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`kbdim`	integer	in	maximum length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`klev`	integer	in	number of model levels (layers)
`klevp1`	integer	in	number of layers plus one
`ktrac`	integer	in	number of tracers
`krow`	integer	in	index number of block of geographical longitudes
`papm1(kbdim,klev)`	double prec.	in	pressure of dry air at center of model layers at time step $t - \Delta t$
`paphm1(kbdim,klevp1)`	double prec.	in	pressure of dry air at interfaces between model layers at time step $t - \Delta t$
`ptm1(kbdim,klev)`	double prec.	in	temperature at center of model layers at time step $t - \Delta t$
`ptte(kbdim,klev)`	double prec.	in	temperature tendency at center of model layers accumulated over all processes of actual time step until call of this subroutine
`pxtm1(kbdim,klev,ktrac)`	double prec.	inout	tracer mass or molar mixing ratio with respect to dry air at center of model layers at time step $t - \Delta t$
`pxtte(kbdim,klev,ktrac)`	double prec.	inout	tendency of tracer mass or molar mixing ratio with respect to dry air at center of model layers accumulated over all processes of actual time step until call of this subroutine

table continued on next page

Table 3.4: `Parameters of physc_subm_1` — continued

`pqm1(kbdim,klev)`	double prec.	in	specific humidity (with respect to dry air) at center of model layers at time step $t - \Delta t$
`pqte(kbdim,klev)`	double prec.	in	tendency of specific humidity (with respect to dry air) at center of model layers accumulated over all processes of actual time step until call of this subroutine

3.4.2.5 Interface of `radiation_subm_1`

Listing 3.15: radiation_subm_1

```
SUBROUTINE radiation_subm_1 &
   (kproma            ,kbdim             ,klev          ,krow   ,&
    ktrac             ,kaero             ,kpband        ,kb_sw  ,&
    aer_tau_sw_vr     ,aer_piz_sw_vr     ,aer_cg_sw_vr          ,&
    aer_tau_lw_vr                                               ,&
    ppd_hl            ,pxtm1                                     )
  INTEGER, INTENT(in) :: kproma
  INTEGER, INTENT(in) :: kbdim
  INTEGER, INTENT(in) :: klev
  INTEGER, INTENT(in) :: krow
  INTEGER, INTENT(in) :: ktrac
  INTEGER, INTENT(in) :: kaero
  INTEGER, INTENT(in) :: kpband
  INTEGER, INTENT(in) :: kb_sw
  REAL(dp), INTENT(inout) :: aer_tau_sw_vr(kbdim,klev,kb_sw), &
                             aer_piz_sw_vr(kbdim,klev,kb_sw), &
                             aer_cg_sw_vr(kbdim,klev,kb_sw),  &
                             aer_tau_lw_vr(kbdim,klev,kpband),&
  REAL(dp), INTENT(in):: ppd_hl(kbdim,klev)
  REAL(dp), INTENT(in):: pxtm1(kbdim,klev,ktrac)
```

Table 3.5: Parameter list of arguments passed to `radiation_subm_1`

name	type	intent	description
`kproma`	integer	in	actual length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`kbdim`	integer	in	maximum length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`klev`	integer	in	number of model levels (layers)

table continued on next page

Table 3.5: `Parameters of radiation_subm_1` — continued

`krow`	integer	in	index number of block of geographical longitudes
`ktrac`	integer	in	number of tracers
`kaero`	integer	in	switch for aerosol radiation coupling
`kpband`	integer	in	number of bands in the thermal radiation wavelength range
`kb_sw`	integer	in	number of bands in the solar radiation wavelength range
`aer_tau_sw_vr (kbdim,klev,kb_sw)`	double prec.	inout	aerosol optical depth of model layers for solar radiation wavelength bands. Here, the model layers are ordered from the Earth's surface (level index 1) to the top of the atmosphere (level index `klev`) as indicated by `_vr` = vertically reversed
`aer_piz_sw_vr (kbdim,klev,kb_sw)`	double prec.	inout	aerosol single scattering albedo for solar radiation wavelength bands. Here, the model layers are ordered from the Earth's surface (level index 1) to the top of the atmosphere (level index `klev`) as indicated by `_vr` = vertically reversed
`aer_cg_sw_vr (kbdim,klev,kb_sw)`	double prec.	inout	aerosol asymmetry factor for solar radiation wavelength bands. Here, the model layers are ordered from the Earth's surface (level index 1) to the top of the atmosphere (level index `klev`) as indicated by `_vr` = vertically reversed
`aer_tau_lw_vr (kbdim,klev,kpband)`	double prec.	inout	aerosol optical depth of model layers for thermal radiation wavelength bands. Here, the model layers are ordered from the Earth's surface (level index 1) to the top of the atmosphere (level index `klev`) as indicated by `_vr` = vertically reversed
`ppd_hl(kbdim,klev)`	double prec.	in	absolute value of dry air pressure difference between upper and lower limit of model layers at time $t - \Delta t$
`pxtm1(kbdim,klev,ktrac)`	double prec.	in	tracer mass or molar mixing ratio with respect to dry air at center of model layers at time step $t - \Delta t$

3.4.2.6 Interface of `radiation_subm_2`

Listing 3.16: radiation_subm_2

```
SUBROUTINE radiation_subm_2(kproma, kbdim, krow, klev, &
                            ktrac, kaero,              &
                            pxtm1                      )
  INTEGER,  INTENT(in) :: kproma
  INTEGER,  INTENT(in) :: kbdim
  INTEGER,  INTENT(in) :: krow
  INTEGER,  INTENT(in) :: klev
  INTEGER,  INTENT(in) :: ktrac
  INTEGER,  INTENT(in) :: kaero
  REAL(dp), INTENT(in) :: pxtm1 (kbdim,klev,ktrac)
```

Table 3.6: Parameter list of arguments passed to `radiation_subm_2`

name	type	intent	description
`kproma`	integer	in	actual length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`kbdim`	integer	in	maximum length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`krow`	integer	in	index number of block of geographical
`klev`	integer	in	number of model levels (layers)
`ktrac`	integer	in	number of tracers
`kaero`	integer	in	switch for aerosol radiation coupling
`pxtm1(kbdim,klev,ktrac)`	double prec.	in	tracer mass or molar mixing ratio with respect to dry air at center of model layers at time step $t - \Delta t$

3.4.2.7 Interface of `vdiff_subm`

Listing 3.17: vdiff_subm

```
SUBROUTINE vdiff_subm(kproma, kbdim,  klev,   klevp1,        &
                      ktrac,  krow,                          &
                      ptm1,   pum1,   pvm1,   pqm1,          &
                      papm1,  paphm1, paphp1, pgeom1, ptslm1,&
                      pxtm1,  pseaice,pforest,               &
                      pfrl,   pfrw,   pfri,   pcvs,   pcvw,  &
                      pvgrat, ptsw,   ptsi,                  &
                      pu10,   pv10,                          &
                      paz0,   paz0l,  paz0w,  paz0i,         &
                      pcfm,   pcfnc,  pepdu2, pkap,          &
                      pri,    ptvir1, ptvl,                  &
                      psrfl,  pcdn,  pqss,   pvlt,           &
                      loland,                                &
```

```
                    pxtte,  pxtems,                              &
                    pxlm1,  pxim1                                )
INTEGER,  INTENT(in) :: kproma
INTEGER,  INTENT(in) :: kbdim
INTEGER,  INTENT(in) :: klev
INTEGER,  INTENT(in) :: klevp1
INTEGER,  INTENT(in) :: ktrac
INTEGER,  INTENT(in) :: krow
REAL(dp), INTENT(in) :: ptm1     (kbdim,klev)
REAL(dp), INTENT(in) :: pum1     (kbdim,klev)
REAL(dp), INTENT(in) :: pvm1     (kbdim,klev)
REAL(dp), INTENT(in) :: pqm1     (kbdim,klev)
REAL(dp), INTENT(in) :: papm1    (kbdim,klev)
REAL(dp), INTENT(in) :: paphm1   (kbdim,klev+1)
REAL(dp), INTENT(in) :: paphp1   (kbdim,klev+1)
REAL(dp), INTENT(in) :: pgeom1   (kbdim,klev)
REAL(dp), INTENT(in) :: ptslm1   (kbdim)
REAL(dp), INTENT(inout) :: pxtm1    (kbdim,klev,ktrac)
REAL(dp), INTENT(in) :: pseaice  (kbdim)
REAL(dp), INTENT(in) :: pforest  (kbdim)
REAL(dp), INTENT(in) :: pfrl     (kbdim)
REAL(dp), INTENT(in) :: pfrw     (kbdim)
REAL(dp), INTENT(in) :: pfri     (kbdim)
REAL(dp), INTENT(in) :: pcvs     (kbdim)
REAL(dp), INTENT(in) :: pcvw     (kbdim)
REAL(dp), INTENT(in) :: pvgrat   (kbdim)
REAL(dp), INTENT(in) :: ptsw     (kbdim)
REAL(dp), INTENT(in) :: ptsi     (kbdim)
REAL(dp), INTENT(in) :: pu10     (kbdim)
REAL(dp), INTENT(in) :: pv10     (kbdim)
REAL(dp), INTENT(in) :: paz0     (kbdim)
REAL(dp), INTENT(in) :: paz0l    (kbdim)
REAL(dp), INTENT(in) :: paz0w    (kbdim)
REAL(dp), INTENT(in) :: paz0i    (kbdim)
REAL(dp), INTENT(in) :: pcfm     (kbdim,klev)
REAL(dp), INTENT(in) :: pcfnc    (kbdim)
REAL(dp), INTENT(in) :: pepdu2
REAL(dp), INTENT(in) :: pkap
REAL(dp), INTENT(in) :: pri      (kbdim)
REAL(dp), INTENT(in) :: ptvir1   (kbdim,klev)
REAL(dp), INTENT(in) :: ptvl     (kbdim)
REAL(dp), INTENT(in) :: psrfl    (kbdim)
REAL(dp), INTENT(in) :: pcdn     (kbdim)
REAL(dp), INTENT(in) :: pqss     (kbdim,klev)
REAL(dp), INTENT(in) :: pvlt     (kbdim)
LOGICAL,  INTENT(in) :: loland   (kbdim)
REAL(dp), INTENT(inout) :: pxtte    (kbdim,klev,ktrac)
REAL(dp), INTENT(inout) :: pxtems   (kbdim,ktrac)
```

```
REAL(dp), INTENT(in) :: pxlm1     (kbdim,klev)
REAL(dp), INTENT(in) :: pxim1     (kbdim,klev)
```

Table 3.7: Parameter list of arguments passed to `vdiff_subm`

name	type	intent	description
`kproma`	integer	in	actual length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`kbdim`	integer	in	maximum length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`klev`	integer	in	number of model levels (layers)
`klevp1`	integer	in	number of layers plus one
`ktrac`	integer	in	number of tracers
`krow`	integer	in	index number of block of geographical longitudes
`ptm1(kbdim,klev)`	double prec.	in	temperature at center of model layers at time step $t - \Delta t$
`pum1(kbdim,klev)`	double prec.	in	zonal wind component at center of model layers at time step $t - \Delta t$
`pvm1(kbdim,klev)`	double prec.	in	meridional wind component at center of model layers at time step $t - \Delta t$
`pqm1(kbdim,klev)`	double prec.	in	specific humidity (with respect to dry air) at center of model layers at time step $t - \Delta t$
`papm1(kbdim,klev)`	double prec.	in	pressure of dry air at center of model layers at time step $t - \Delta t$
`paphm1(kbdim,klevp1)`	double prec.	in	pressure of dry air at interfaces between model layers at time step $t - \Delta t$
`paphp1(kbdim,klevp1)`	double prec.	in	pressure of dry air at interfaces between model layers at prognostic time step $t + \Delta t$
`pgeom1(kbdim,klev)`	double prec.	in	geopotential at center of model layers at time step $t - \Delta t$
`ptslm1(kbdim)`	double prec.	in	surface temperature at time step $t - \Delta t$
`pxtm1(kbdim,klev,ktrac)`	double prec.	inout	tracer mass or molar mixing ratio with respect to dry air at center of model layers at time step $t - \Delta t$
`pseaice(kbdim)`	double prec.	in	sea ice fraction
`pforest(kbdim)`	double prec.	in	forest fraction
`pfrl(kbdim)`	double prec.	in	land fraction
`pfrw(kbdim)`	double prec.	in	surface water fraction
`pfri(kbdim)`	double prec.	in	surface ice fraction

table continued on next page

Table 3.7: `Parameters of vdiff_subm` — continued

`pcvs(kbdim)`	double prec.	in	snow cover fraction
`pcvw(kbdim)`	double prec.	in	wet skin fraction
`pvgrat(kbdim)`	double prec.	in	vegetation ratio
`ptsw(kbdim)`	double prec.	in	surface temperature over water
`ptsi(kbdim)`	double prec.	in	surface temperature over ice
`pu10(kbdim)`	double prec.	in	zonal wind component 10 m above the surface
`pv10(kbdim)`	double prec.	in	meridional wind component 10 m above the surface
`paz0(kbdim)`	double prec.	in	roughness length
`paz0l(kbdim)`	double prec.	in	roughness length over land
`paz0w(kbdim)`	double prec.	in	roughness length over water
`paz0i(kbdim)`	double prec.	in	roughness length over ice
`pcfm(kbdim,klev)`	double prec.	in	stability dependent momentum transfer coefficient at center of model layers
`pcfnc(kbdim)`	double prec.	in	function of heat transfer coefficient; not set?
`pepdu2`	double prec.	in	a constant set in `vdiff.f90`. It is used e.g. in `mo_surface_land` as the allowed minimum of the square of the absolute wind velocity
`pkap`	double prec.	in	von Karman constant
`pri(kbdim)`	double prec.	in	Richardson number for moist air
`ptvir1(kbdim,klev)`	double prec.	in	potential density temperature
`ptvl(kbdim)`	double prec.	in	virtual temperature over land
`psrfl(kbdim)`	double prec.	in	net surface solar radiation flux at time (?) t
`pcdn(kbdim)`	double prec.	in	heat transfer coefficient averaged over land, water and ice cover fraction of a grid box
`pqss(kbdim,klev)`	double prec.	in	specific humidity at which the air is saturated at time (?) t
`pvlt(kbdim)`	double prec.	in	obsolete, will be removed
`loland(kbdim)`	double prec.	in	logical land mask including glaciers
`pxtte(kbdim,klev,ktrac)`	double prec.	inout	tendency of tracer mass or molar mixing ratio with respect to dry air at center of model layers accumulated over all processes of actual time step until call of this subroutine
`pxtems(kbdim,ktrac)`	double prec.	inout	surface emission flux
`pxlm1`	double prec.	in	cloud liquid water content at center of model layers at time step $t - \Delta t$
`pxim1`	double prec.	in	cloud water ice content at center of model layers at time step $t - \Delta t$

3.4.2.8 Interface of `rad_heat_subm`

Listing 3.18: rad_heat_subm

```
SUBROUTINE radheat_subm
   (kproma        ,kbdim            ,klev              ,&
    klevp1        ,krow             ,pconvfact         ,&
    pflxs         ,pflxt)

  INTEGER, INTENT(in)  :: kproma
  INTEGER, INTENT(in)  :: kbdim
  INTEGER, INTENT(in)  :: klev
  INTEGER, INTENT(in)  :: klevp1
  INTEGER, INTENT(in)  :: krow
  REAL(dp), INTENT(in) :: pconvfact(kbdim,klev)
  REAL(dp), INTENT(in) :: pflxs(kbdim,klevp1), pflxt(kbdim,klevp1
     )
```

Table 3.8: Parameter list of arguments passed to `rad_heat_subm`

name	type	intent	description
`kproma`	integer	in	actual length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`kbdim`	integer	in	maximum length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`klev`	integer	in	number of model levels (layers)
`klevp1`	integer	in	number of layers plus one
`krow`	integer	in	index number of block of geographical longitudes
`pconvfact(kbdim,klevp1)`	double prec.	in	conversion factor for conversion of energy flux differences between upper and lower layer boundary to heating rate of the air in this layer. The factor is calculated for the time at time step $t-\Delta t$.
`pflxs(kbdim,klevp1)`	double prec.	in	net energy flux of solar radiation integrated over all solar radiation bands at the layer interfaces for time t
`pflxt(kbdim,klevp1)`	double prec.	in	net energy flux of thermal radiation integrated over all thermal radiation bands at the layer interfaces for time t

3.4.2.9 Interface of `physc_subm_2`

Listing 3.19: physc_subm_2

```
SUBROUTINE physc_subm_2                                    &
          (kproma, kbdim, klev,  klevp1, ktrac, krow, &
           itrpwmo, itrpwmop1,                             &
           paphm1, papm1, paphp1, papp1,                   &
           ptm1,   ptte,  ptsurf,                          &
           pqm1,   pqte,  pxlm1, pxlte, pxim1, pxite, &
           pxtm1,  pxtte,                                  &
           paclc,  ppbl,                                   &
           loland, loglac                                 )
  INTEGER, INTENT(in)  :: kproma
  INTEGER, INTENT(in)  :: kbdim
  INTEGER, INTENT(in)  :: klev
  INTEGER, INTENT(in)  :: klevp1
  INTEGER, INTENT(in)  :: ktrac
  INTEGER,  INTENT(in) :: krow
  INTEGER,  INTENT(in) :: itrpwmo  (kbdim)
  INTEGER,  INTENT(in) :: itrpwmop1(kbdim)

  REAL(dp), INTENT(in) :: paphm1   (kbdim,klev+1)
  REAL(dp), INTENT(in) :: papm1    (kbdim,klev)
  REAL(dp), INTENT(in) :: paphp1   (kbdim,klev+1)
  REAL(dp), INTENT(in) :: papp1    (kbdim,klev)
  REAL(dp), INTENT(in) :: ptm1     (kbdim,klev)
  REAL(dp), INTENT(in) :: ptte     (kbdim,klev)
  REAL(dp), INTENT(in) :: ptsurf   (kbdim)
  REAL(dp), INTENT(in) :: pqm1     (kbdim,klev)
  REAL(dp), INTENT(in) :: pqte     (kbdim,klev)
  REAL(dp), INTENT(in) :: pxlm1    (kbdim,klev)
  REAL(dp), INTENT(in) :: pxlte    (kbdim,klev)
  REAL(dp), INTENT(in) :: pxim1    (kbdim,klev)
  REAL(dp), INTENT(in) :: pxite    (kbdim,klev)
  REAL(dp), INTENT(in) :: paclc    (kbdim,klev)
  REAL(dp), INTENT(in) :: ppbl     (kbdim)
  REAL(dp), INTENT(inout) :: pxtm1 (kbdim,klev,ktrac)
  REAL(dp), INTENT(inout) :: pxtte (kbdim,klev,ktrac)
  LOGICAL,  INTENT(in) :: loland   (kbdim)
  LOGICAL,  INTENT(in) :: loglac   (kbdim)
```

Table 3.9: Parameter list of arguments passed to `physc_subm_2`

name	type	intent	description
`kproma`	integer	in	actual length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)

table continued on next page

Table 3.9: `Parameters of physc_subm_2` — continued

`kbdim`	integer	in	maximum length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`klev`	integer	in	number of model levels (layers)
`klevp1`	integer	in	number of layers plus one
`ktrac`	integer	in	number of tracers
`krow`	integer	in	index number of block of geographical longitudes
`itrpwmo(kbdim)`	integer	in	index of model level at which meteorological tropopause was detected at time t
`itrpwmop1(kbdim)`	integer	in	index of model level at which meteorological tropopause was detected plus 1 at time t
`paphm1(kbdim,klevp1)`	double prec.	in	pressure of dry air at interfaces between model layers at time step $t - \Delta t$
`papm1(kbdim,klev)`	double prec.	in	pressure of dry air at center of model layers at time step $t - \Delta t$
`paphp1(kbdim,klevp1)`	double prec.	in	pressure of dry air at interfaces between model layers at prognostic time step $t + \Delta t$
`papp1(kbdim,klev)`	double prec.	in	pressure of dry air at center of model layers at time step $t + \Delta t$
`ptm1(kbdim,klev)`	double prec.	in	temperature at center of model layers at time step $t - \Delta t$
`ptte(kbdim,klev)`	double prec.	in	temperature tendency at center of model layers accumulated over all processes of actual time step until call of this subroutine
`ptsurf(kbdim)`	double prec.	in	surface temperature at time step t
`pqm1(kbdim,klev)`	double prec.	in	specific humidity (with respect to dry air) at center of model layers at time step $t - \Delta t$
`pqte(kbdim,klev)`	double prec.	in	tendency of specific humidity (with respect to dry air) at center of model layers accumulated over all processes of actual time step until call of this subroutine
`pxlm1`	double prec.	in	cloud liquid water content at center of model layers at time step $t - \Delta t$
`pxlte`	double prec.	in	cloud liquid water tendency at center of model layers accumulated over all processes of actual time step until call of this subroutine

table continued on next page

Table 3.9: `Parameters of physc_subm_2` — continued

`pxim1`	double prec.	in	cloud water ice content at center of model layers at time step $t - \Delta t$
`pxite`	double prec.	in	cloud water ice tendency at center of model layers accumulated over all processes of actual time step until call of this subroutine
`pxtm1(kbdim,klev,ktrac)`	double prec.	inout	tracer mass or molar mixing ratio with respect to dry air at center of model layers at time step $t - \Delta t$
`pxtte(kbdim,klev,ktrac)`	double prec.	inout	tendency of tracer mass or molar mixing ratio with respect to dry air at center of model layers accumulated over all processes of actual time step until call of this subroutine
`paclc(kbdim,klev)`	double prec.	in	cloud fraction at center of model layers at time step t
`ppbl(kbdim)`	double prec.	in	model layer index of geometrically highest model layer of planetary boundary layer converted to a real number at time t
`loland(kbdim)`	double prec.	in	logical land mask including glaciers
`loglac(kbdim)`	double prec.	in	logical glacier mask

3.4.2.10 Interface of `cuflx_subm`

Listing 3.20: cuflx_subm

```
SUBROUTINE cuflx_subm(kbdim,  kproma,  klev,    ktop,   krow, &
                      pxtenh, pxtu,    prhou,                 &
                      pmfu,   pmfuxt,                         &
                      pmlwc,  pmiwc,   pmratepr,pmrateps,     &
                      pfrain, pfsnow,  pfevapr, pfsubls,      &
                      paclc,  pmsnowacl,                      &
                      ptu,    pdpg,                           &
                      pxtte                                   )
  INTEGER, INTENT(in)     :: kbdim, kproma, klev, ktop, &
                             krow
  REAL(dp), INTENT(in)    :: pdpg(kbdim,klev),          &
                             pmratepr(kbdim,klev),      &
                             pmrateps(kbdim,klev),      &
                             pmsnowacl(kbdim,klev),     &
                             ptu(kbdim,klev),           &
                             pfrain(kbdim,klev),        &
                             pfsnow(kbdim,klev),        &
                             pfevapr(kbdim,klev),       &
                             pfsubls(kbdim,klev),       &
                             pmfu(kbdim,klev),          &
```

```
                                paclc(kbdim,klev),          &
                                prhou(kbdim,klev)
 REAL(dp), INTENT(inout) :: pxtte(kbdim,klev,ntrac),    &
                            pmlwc(kbdim,klev),          &
                            pmiwc(kbdim,klev),          &
                            pxtenh(kbdim,klev,ntrac),   &
                            pxtu(kbdim,klev,ntrac),     &
                            pmfuxt(kbdim,klev,ntrac)
```

Table 3.10: Parameter list of arguments passed to `cuflx_subm`

name	type	intent	description
`kbdim`	integer	in	maximum length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`kproma`	integer	in	actual length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`klev`	integer	in	number of model levels (layers)
`ktop`	integer	in	Could be the minimum model layer index of cloud top layers over one block. In fact, it is set to 1 in `cuflx`
`pxtenh(kbdim,klev,ntrac)`	double prec.	inout	tracer mass or molar mixing ratio with respect to dry air at center of model layers at time step $t + \Delta t$
`pxtu(kbdim,klev,ntrac)`	double prec.	inout	tracer mass mixing ratio with respect to cloud water at center of model layers in the liquid or solid cloud water phase at time step $t + \Delta t$
`prhou(kbdim,klev)`	double prec.	in	dry air density at center of model layers at time step $t + \Delta t$
`pmfu(kbdim,klev)`	double prec.	in	convective air mass flux at center of model layers at time t
`pmfuxt(kbdim,klev,ntrac)`	double prec.	inout	net tracer mass flux due to convective transport and wet deposition at center of model layers at time step $t + \Delta t$ on exit (in mass mixing ratio per time)
`pmlwc(kbdim,klev)`	double prec.	inout	liquid water content (mass of liquid water per mass of dry air) at center of model layers at time $t + \Delta t$ on exit
`pmiwc(kbdim,klev)`	double prec.	inout	ice water content (mass of water ice per mass of dry air) at center of model layers at time $t + \Delta t$ on exit

table continued on next page

Table 3.10: `Parameters of cuflx_subm` — continued

`pmratepr(kbdim,klev)`	double prec.	in	rain formation rate in mass water per mass dry air converted to rain at center of model layers at time step t
`pmrateps(kbdim,klev)`	double prec.	in	ice formation rate in mass water per mass dry air converted to snow at center of model layers at time step t
`pfrain(kbdim,klev)`	double prec.	in	rain flux at centers of model layers per grid box area at time t, evaporation not taken into account
`pfsnow(kbdim,klev)`	double prec.	in	snow flux at centers of model layers per grid box area at time t, evaporation not taken into account
`pfevapr(kbdim,klev)`	double prec.	in	evaporation of rain at centers of model layers per grid box area at time t
`pfsubls(kbdim,klev)`	double prec.	in	sublimation of snow at centers of model layers per grid box area at time t
`paclc(kbdim,klev)`	double prec.	in	cloud cover at center of model layer at time step t
`pmsnowaclc(kbdim,klev)`	double prec.	in	accretion rate of snow at center of model layer at time step t
`ptu(kbdim,klev)`	double prec.	in	temperature at center of model layer at time step $t - \Delta t$
`pdpg(kbdim,klev)`	double prec.	in	geopotential height at center of model level
`pxtte(kbdim,klev,ktrac)`	double prec.	inout	tendency of tracer mass or molar mixing ratio with respect to dry air at center of model layers accumulated over all processes of actual time step until call of this subroutine

3.4.2.11 Interface of `cloud_subm`

Listing 3.21: cloud_subm

```
SUBROUTINE cloud_subm(                                          &
           kproma,      kbdim,       klev,        ktop,       &
           krow,                                                &
           pmlwc,       pmiwc,       pmratepr,    pmrateps,   &
           pfrain,      pfsnow,      pfevapr,     pfsubls,    &
           pmsnowacl,   paclc,       ptm1,        ptte,       &
           pxtm1,       pxtte,       paphp1,      papp1,      &
           prhop1,      pclcpre)

  INTEGER,  INTENT(in)    :: kproma
  INTEGER,  INTENT(in)    :: kbdim
  INTEGER,  INTENT(in)    :: klev
  INTEGER,  INTENT(in)    :: ktop
```

```
 INTEGER,  INTENT(in)    :: krow
 REAL(dp), INTENT(in)    :: pclcpre  (kbdim,klev)
 REAL(dp), INTENT(in)    :: pfrain   (kbdim,klev)
 REAL(dp), INTENT(in)    :: pfsnow   (kbdim,klev)
 REAL(dp), INTENT(in)    :: pfevapr  (kbdim,klev)
 REAL(dp), INTENT(in)    :: pfsubls  (kbdim,klev)
 REAL(dp), INTENT(in)    :: pmsnowacl(kbdim,klev)
 REAL(dp), INTENT(in)    :: ptm1     (kbdim,klev)
 REAL(dp), INTENT(in)    :: ptte     (kbdim,klev)
 REAL(dp), INTENT(in)    :: prhop1   (kbdim,klev)
 REAL(dp), INTENT(in)    :: papp1    (kbdim,klev)
 REAL(dp), INTENT(in)    :: paphp1   (kbdim,klev+1)
 REAL(dp), INTENT(inout) :: paclc    (kbdim,klev)
 REAL(dp), INTENT(inout) :: pmlwc    (kbdim,klev)
 REAL(dp), INTENT(inout) :: pmiwc    (kbdim,klev)
 REAL(dp), INTENT(inout) :: pmratepr (kbdim,klev)
 REAL(dp), INTENT(inout) :: pmrateps (kbdim,klev)
 REAL(dp), INTENT(in)    :: pxtm1    (kbdim,klev,ntrac)
 REAL(dp), INTENT(inout) :: pxtte    (kbdim,klev,ntrac)
```

Table 3.11: Parameter list of arguments passed to `cloud_subm`

name	type	intent	description
`kproma`	integer	in	actual length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`kbdim`	integer	in	maximum length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`klev`	integer	in	number of model levels (layers)
`ktop`	integer	in	Could be the minimum model layer index of cloud top layers over one block. In fact, it is set to 1 in `cuflx`
`krow`	integer	in	index number of block of geographical longitudes
`pmlwc(kbdim,klev)`	double prec.	inout	liquid water content (mass of liquid water per mass of dry air) at center of model layers at time $t + \Delta t$ on exit
`pmiwc(kbdim,klev)`	double prec.	inout	ice water content (mass of water ice per mass of dry air) at center of model layers at time $t + \Delta t$ on exit
`pmratepr(kbdim,klev)`	double prec.	inout	rain formation rate in mass water per mass dry air converted to rain at center of model layers at time step t

table continued on next page

Table 3.11: `Parameters of cloud_subm` — continued

`pmrateps(kbdim,klev)`	double prec.	inout	ice formation rate in mass water per mass dry air converted to snow at center of model layers at time step t
`pfrain(kbdim,klev)`	double prec.	in	rain flux at centers of model layers per grid box area at time t, evaporation not taken into account
`pfsnow(kbdim,klev)`	double prec.	in	snow flux at centers of model layers per grid box area at time t, evaporation not taken into account
`pfevapr(kbdim,klev)`	double prec.	in	evaporation of rain at centers of model layers per grid box area at time t
`pfsubls(kbdim,klev)`	double prec.	in	sublimation of snow at centers of model layers per grid box area at time t
`pmsnowaclc(kbdim,klev)`	double prec.	in	accretion rate of snow at center of model layer at time step t
`paclc(kbdim,klev)`	double prec.	inout	cloud cover at center of model layer at time step t
`ptm1(kbdim,klev)`	double prec.	in	temperature at center of model layers at time step $t - \Delta t$
`ptte(kbdim,klev)`	double prec.	in	temperature tendency at center of model layers accumulated over all processes of actual time step until call of this subroutine
`pxtm1(kbdim,klev,ntrac)`	double prec.	in	tracer mass or molar mixing ratio with respect to dry air at center of model layers at time step $t - \Delta t$
`pxtte(kbdim,klev,ntrac)`	double prec.	inout	tendency of tracer mass or molar mixing ratio with respect to dry air at center of model layers accumulated over all processes of actual time step until call of this subroutine
`paphp1(kbdim,klev+1)`	double prec.	in	pressure of dry air at interfaces between model layers at prognostic time step $t+\Delta t$
`papp1(kbdim,klev)`	double prec.	in	pressure of dry air at center of model layers at time step $t + \Delta t$
`prhop1(kbdim,klev)`	double prec.	in	dry air density at center of model layers at time step $t + \Delta t$
`pclcpre(kbdim,klev)`	double prec.	in	fraction of grid box covered by precipitation at time step t

3.4.2.12 Interface of `physc_subm_3`

Listing 3.22: physc_subm_3

```
SUBROUTINE physc_subm_3                                    &
          (kproma, kbdim, klev,  klevp1, ktrac, krow, &
```

```
          paphm1, papm1, paphp1, papp1,                &
          ptm1,   ptte,  ptsurf,                       &
          pqm1,   pqte,                                &
          pxlm1,  pxlte, pxim1, pxite,                 &
          pxtm1,  pxtte,                               &
          pgeom1, pgeohm1,                             &
          paclc,                                       &
          ppbl,   pvervel,                             &
          loland, loglac                               )
 INTEGER,  INTENT(in)    :: kproma
 INTEGER,  INTENT(in)    :: kbdim
 INTEGER,  INTENT(in)    :: klev
 INTEGER,  INTENT(in)    :: klevp1
 INTEGER,  INTENT(in)    :: ktrac
 INTEGER,  INTENT(in)    :: krow
 REAL(dp), INTENT(in)    :: paphm1   (kbdim,klevp1)
 REAL(dp), INTENT(in)    :: papm1    (kbdim,klev)
 REAL(dp), INTENT(in)    :: paphp1   (kbdim,klevp1)
 REAL(dp), INTENT(in)    :: papp1    (kbdim,klev)
 REAL(dp), INTENT(in)    :: ptm1     (kbdim,klev)
 REAL(dp), INTENT(inout) :: ptte     (kbdim,klev)
 REAL(dp), INTENT(in)    :: ptsurf   (kbdim)
 REAL(dp), INTENT(in)    :: pqm1     (kbdim,klev)
 REAL(dp), INTENT(inout) :: pqte     (kbdim,klev)
 REAL(dp), INTENT(in)    :: pxlm1    (kbdim,klev)
 REAL(dp), INTENT(inout) :: pxlte    (kbdim,klev)
 REAL(dp), INTENT(in)    :: pxim1    (kbdim,klev)
 REAL(dp), INTENT(inout) :: pxite    (kbdim,klev)
 REAL(dp), INTENT(inout) :: pxtm1    (kbdim,klev,ktrac)
 REAL(dp), INTENT(inout) :: pxtte    (kbdim,klev,ktrac)
 REAL(dp), INTENT(in)    :: pgeom1   (kbdim,klev)
 REAL(dp), INTENT(in)    :: pgeohm1  (kbdim,klevp1)
 REAL(dp), INTENT(in)    :: paclc    (kbdim,klev)
 REAL(dp), INTENT(in)    :: ppbl     (kbdim)
 REAL(dp), INTENT(in)    :: pvervel  (kbdim,klev)
 LOGICAL,  INTENT(in)    :: loland   (kbdim)
 LOGICAL,  INTENT(in)    :: loglac   (kbdim)
```

Table 3.12: Parameter list of arguments passed to `physc_subm_3`

name	type	intent	description
`kproma`	integer	in	actual length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)

table continued on next page

Table 3.12: `Parameters of physc_subm_3` — continued

`kbdim`	integer	in	maximum length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`klev`	integer	in	number of model levels (layers)
`klevp1`	integer	in	number of layers plus one
`ktrac`	integer	in	number of tracers
`krow`	integer	in	index number of block of geographical longitudes
`paphm1(kbdim,klevp1)`	double prec.	in	pressure of dry air at interfaces between model layers at time step $t - \Delta t$
`papm1(kbdim,klev)`	double prec.	in	pressure of dry air at center of model layers at time step $t - \Delta t$
`paphp1(kbdim,klevp1)`	double prec.	in	pressure of dry air at interfaces between model layers at prognostic time step $t + \Delta t$
`papp1(kbdim,klev)`	double prec.	in	pressure of dry air at center of model layers at time step $t + \Delta t$
`ptm1(kbdim,klev)`	double prec.	in	temperature at center of model layers at time step $t - \Delta t$
`ptte(kbdim,klev)`	double prec.	inout	temperature tendency at center of model layers accumulated over all processes of actual time step until call of this subroutine
`ptsurf(kbdim)`	double prec.	in	surface temperature at time step t
`pqm1(kbdim,klev)`	double prec.	in	specific humidity (with respect to dry air) at center of model layers at time step $t - \Delta t$
`pqte(kbdim,klev)`	double prec.	inout	tendency of specific humidity (with respect to dry air) at center of model layers accumulated over all processes of actual time step until call of this subroutine
`pxlm1`	double prec.	in	cloud liquid water content (mass of liquid water per mass of dry air) at center of model layers at time step $t - \Delta t$
`pxlte`	double prec.	inout	cloud liquid water tendency (rate of change of mass of liquid water per mass of dry air) at center of model layers accumulated over all processes of actual time step until call of this subroutine
`pxim1`	double prec.	in	cloud water ice content (mass of water ice per mass of dry air) at center of model layers at time step $t - \Delta t$

table continued on next page

Table 3.12: `Parameters of physc_subm_3` — continued

`pxite`	double prec.	inout	cloud water ice tendency (rate of change of mass of ice water per mass of dry air) at center of model layers accumulated over all processes of actual time step until call of this subroutine
`pxtm1(kbdim,klev,ktrac)`	double prec.	inout	tracer mass or molar mixing ratio with respect to dry air at center of model layers at time step $t - \Delta t$
`pxtte(kbdim,klev,ktrac)`	double prec.	inout	tendency of tracer mass or molar mixing ratio with respect to dry air at center of model layers accumulated over all processes of actual time step until call of this subroutine
`pgeom1(kbdim,klev)`	double prec.	in	geopotential at center of model layers at time step $t - \Delta t$
`pgeohm1(kbdim,klevp1)`	double prec.	in	geopotential at interfaces between model layers at time step $t - \Delta t$
`paclc(kbdim,klev)`	double prec.	in	cloud cover at center of model layer at time step t
`ppbl(kbdim)`	double prec.	in	model layer index of geometrically highest model layer of planetary boundary layer converted to a real number at time t
`pvervel(kbdim,klev)`	double prec.	in	large scale vertical velocity at model center at time step t
`loland(kbdim)`	double prec.	in	logical land mask including glaciers
`loglac(kbdim)`	double prec.	in	logical glacier mask

3.4.2.13 Interface of `physc_subm_4`

Listing 3.23: physc_subm_4

```
SUBROUTINE physc_subm_4 (kproma,  kbdim,   klev,            &
                         klevp1,  ktrac,   krow,            &
                         paphm1,  pfrl,    pfrw,            &
                         pfri,    loland,  pxtm1,           &
                         pxtte)
  INTEGER, INTENT(in)   :: kproma, kbdim, klev, klevp1, ktrac,
     krow
  REAL(dp), INTENT(in)  :: paphm1(kbdim,klevp1), &
                           pfrl(kproma),         &
                           pfrw(kproma),         &
                           pfri(kproma),         &
                           pxtm1(kbdim,klev,ktrac)
  REAL(dp), INTENT(inout)::pxtte(kbdim,klev,ktrac)
  LOGICAL, INTENT(in)   :: loland(kproma)
```

Table 3.13: Parameter list of arguments passed to `physc_subm_4`

name	type	intent	description
`kproma`	integer	in	actual length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`kbdim`	integer	in	maximum length of block of geographical longitudes (one longitude block can contain grid cells of various geographical latitudes)
`klev`	integer	in	number of model levels (layers)
`klevp1`	integer	in	number of layers plus one
`ktrac`	integer	in	number of tracers
`krow`	integer	in	index number of block of geographical longitudes
`paphm1(kbdim,klevp1)`	double prec.	in	pressure of dry air at interfaces between model layers at time step $t - \Delta t$
`pfrl(kbdim)`	double prec.	in	land fraction
`pfrw(kbdim)`	double prec.	in	surface water fraction
`pfri(kbdim)`	double prec.	in	surface ice fraction
`loland(kbdim)`	double prec.	in	logical land mask including glaciers
`pxtm1(kbdim,klev,ktrac)`	double prec.	in	tracer mass or molar mixing ratio with respect to dry air at center of model layers at time step $t - \Delta t$
`pxtte(kbdim,klev,ktrac)`	double prec.	inout	tendency of tracer mass or molar mixing ratio with respect to dry air at center of model layers accumulated over all processes of actual time step until call of this subroutine

3.4.2.14 Interface of `free_subm_memory`

Listing 3.24: free_subm_memory

```
SUBROUTINE free_subm_memory
```

This subroutine has no parameter list.

3.4.3 Tracer interface

Tracer fields are constituents transported with the flow of air in the atmospheric model. In addition to the transport, they are subject to several processes such as convection, diffusion, emission, deposition and chemical conversion. Horizontal and vertical transport is carried out by the atmospheric model and some standard processes can be performed by the atmospheric model as well. Other processes which are specific for the tracer must be calculated by the sub–model. The tracer interface is a collection of subroutines that allow the definition and handling of a data structure containing information about tracers. This information comprises the 3–dimensional mass or volume mixing ratio of the tracers but also variables that determine the transport and physical properties of each individual tracer.

Tracers within ECHAM6 are represented by a 4–dimensional array (the three spatial dimensions are supplemented by the tracer index) but pointers to individual tracers can be obtained so that details of implementation of the data structure remains hidden. A one dimensional array of a derived data type holds the meta–information. In the restart file the tracers are identified by name, so that restarts can be continued with different sets of tracers if required. Reading and writing of the tracers to the rerun file and to the output stream is based on the output stream and memory buffer facilities described in section 3.2.

3.4.3.1 Request a new tracer

A new tracer with name `'A'` is requested from a module with name `'my_module'` by a call to the routine `new_tracer` of `mo_tracer.f90`:

```
call new_tracer ('A', 'my_module', idx)
```

Tracer properties are specified by optional arguments of the `new_tracer` subroutine. The interface is as follows:

SUBROUTINE new_tracer		name, modulename [,spid] [,subname] [,trtype] [,idx] [,nwrite] [,longname] [,units] [,moleweight] [,code] [,table] [,bits] [,nbudget] [,burdenid] [,ninit] [,vini] [,nrerun] [,nint] [,ntran] [,nfixtyp] [,nvdiff] [,nconv] [,nwetdep] [,ndrydep] [,nsedi] [,nemis] [,tdecay] [,nphase] [,nsoluble] [,mode] [,myflag] [,ierr])			
name	type	intent	default	function*	description
identification of the tracer :					
name	character(len=*)	in		es	name of the tracer
modulename	character(len=*)	in		es	name of the module requesting the tracer
[spid]	integer	in		es	species index
[subname]	character(len=*)	in		es	optional for 'colored' tracers
[trtype]	integer	in		es	tracer type
[idx]	integer	out		es	index of the tracer
postprocessing output :					
[nwrite]	integer	in	ON	p	flag to print the tracer
[longname]	character(len=*)	in	" "	p	long name
[units]	character(len=*)	in	" "	p	physical units
[moleweight]	real	in	0.	p	molecular weight
[code]	integer	in	0	p	GRIB code
[table]	integer	in	131	p	GRIB table
[bits]	integer	in	16	p	number of GRIB encoding bits
[nbudget]	integer	in	OFF	ep%	budget flag
[burdenid]	integer	in		e%	burden diagnostics number
initialization and rerun :					
[ninit]	integer	in	RESTART+ CONSTANT	e	initialization flag
[vini]	real	in	0.	e	initialization value
[nrerun]	integer	in	ON	e	restart flag
transport and other processes :					
[nint]	integer	in		e	integration flag
[ntran]	integer	in	TRANSPORT	e%	transport switch
[nfixtyp]	integer	in	1	e%	type of mass fixer
[nvdiff]	integer	in	ON	e	vertical diffusion flag
[nconv]	integer	in	ON	e	convection flag
[nwetdep]	integer	in	OFF	e	wet deposition flag
[ndrydep]	integer	in	OFF	e%	dry deposition flag
[nsedi]	integer	in	OFF	e%	sedimentation flag
[nemis]	integer	in	OFF	e%	surface emission flag
[tdecay]	real	in	0.	e	exponential decay time
attributes interpreted by the submodel :					
[nphase]	integer	in	0	s	phase indicator
[nsoluble]	integer	in		s	solubility flag
[mode]	integer	in	0	s	mode indicator
[myflag(:)]	type(t_flag)	in	(",0.)	s	user defined flags
miscellaneous arguments :					
[ierr]	integer	out	OK=0	s	error return value

* attributes interpreted by ECHAM (e), by the submodel (s), by the postprocessing module (p), not yet implemented (%).

In general, integer values are chosen to represent the flags in order to allow different choices:
0: OFF
1: ON, standard action
2: ..., alternative action
...
tag: specific action performed by the sub-model.
Small numbers indicate that some kind of standard action shall be performed by ECHAM. Higher tag values indicate that the process will be handled by the submodel. For the following actual arguments, valid values are defined by parameter statements (see mo_tracdef.f90):

argument	values	description
	OK, OFF, ON	universal values
ntran	NO_ADVECTION, SEMI_LAGRANGIAN, SPITFIRE, TPCORE	transport flag
ninit	INITIAL, RESTART, CONSTANT, PERIODIC	initialization flag
nsoluble	SOLUBLE, INSOLUBLE	soluble flag
nphase	GAS, AEROSOL, GAS_OR_AEROSOL, AEROSOLMASS, AEROSOLNUMBER, UNDEFINED	phase indicator
code	AUTO	automatically chose unique GRIB code
ierr	OK,NAME_USED,NAME_MISS,TABLE_FULL	error return value (cannot be used currently)

Tracer properties: Identification of the tracer and sub-model. Each tracer is identified by a unique name and optionally by a subname in case of colored tracers. In the postprocessing file colored tracers appear with the name name_subname. Values of optional arguments provided for the corresponding non–colored tracer (without argument subname) are used for the colored tracer as well (despite the GRIB code number).
The sub–model identifies itself by a unique character string modulename. idx is the index of the new tracer in the global arrays XT, XTM1, trlist.

Tracer properties: Postprocessing flags. The flag nwrite (default ON) determines, whether the tracer is written to the standard output stream. A separate file with name STANDARDFILENAME_tracer for GRIB, or STANDARDFILENAME_tracer.nc for NetCDF format, is written. The default file format GRIB can be changed to NetCDF by setting trac_filetype=2 in the namelist runctl (see Tab. 2.12 of section 2.2.1.14).
If present, the attributes longname, units and moleweight are written to the NetCDF file.
Within GRIB files, fields are identified by a GRIB code number which must be given as argument code. Note that codes 129 and 152 should not be used because they are attributed to surface pressure and geopotential height. A predefined value AUTO is accepted indicating automatic generation of unique GRIB code numbers. For GRIB files, a code file STANDARDFILENAME_tracer.codes is written to associate code numbers with tracer names. For the tracers, a default GRIB table number 131 is chosen for tracer output. By default, 16 bits are used for encoding in GRIB format.

Tracer properties: Initialization and rerun. The nrerun flag (default=ON) indicates, whether the tracer variable shall be read and written from/to the rerun file. The tracers are identified by name in the rerun (NetCDF) file, so that they can be read selectively. The initialization flag ninit is used to specify the initialization procedure in more detail: Valid values are one of INITIAL (read from initial file, this must be done by the submodel), RESTART

(read from restart file), CONSTANT (set to the initial value `vini`) or a combination (e.g. RESTART+CONSTANT) to indicate that the quantity is read from the restart file in case of a rerun but set to a predefined value otherwise.

Tracer properties: Transport and other processes. Tracer transport and the impact of certain other processes is calculated by ECHAM. The flags `nint`, `ntran`, `nfixtyp`, `nvdiff`, `nconv`, `nwetdep`, `nsedi`, `ndrydep`, `nemis`, `tdecay` are meant to switch `ON` or `OFF` the respective processes (not fully implemented currently).
A value of `tdecay`$\neq 0$ leads to an exponential decay of the tracer with time.

Tracer properties: Attributes interpreted by the submodel. The following flags are not used by ECHAM. They are reserved to be used by the sub-models: `nphase`, `nsoluble`, `mode` and `myflag`. `myflag` is an array of pairs of character strings and real values.

3.4.3.2 Access to tracers with `get_tracer`

The routine `get_tracer` returns the references to tracers already defined.

Example:

Listing 3.25: get_tracer

```
CALL get_tracer ('SO2',idx=index,modulename=modulename)
IF (ierr==0) THEN
  PRINT *, 'Using tracer SO2 from module',modulename
ELSE
  ! eg. read constant tracer field
  ...
ENDIF
```

The interface of subroutine `get_tracer` is:

`SUBROUTINE get_tracer`		`(name [,subname] [,modulename] [,idx] [,pxt] [,pxtm1] [,ierr])`	
name	type	intent	description
name	character(len=*)	in	name of the tracer
[subname]	character(len=*)	in	subname of the tracer
[modulename]	character(len=*)	out	name of requesting module
[idx]	integer	out	index of the tracer
[pxt(:,:,:)]	real	pointer	pointer to the tracer field
[pxtm1(:,:,:)]	real	pointer	pointer to the tracer field at previous time step
[ierr]	integer	out	error return value (0=OK, 1=tracer not defined)

If the optional parameter `ierr` is not given and the tracer is not defined the program will abort. Note that references (`pxt, pxtm1`) to the allocated memory cannot be obtained before all tracers are defined and the respective memory is allocated in the last step of tracer definition.

3.4.3.3 Tracer list data type

Summary information on the tracers is stored in a global variable `trlist`. Attributes of individual tracers are stored in the component array `trlist% ti(:)`. The definitions of the respective

data types t_trlist and t_trinfo are given below:

Listing 3.26: t_trlist

```
!
! Basic data type definition for tracer info list
!
TYPE t_trlist
  !
  ! global tracer list information
  !
  INTEGER          :: ntrac         ! number of tracers specified
  INTEGER          :: anyfixtyp     ! mass fixer types used
  INTEGER          :: anywetdep     ! wet deposition requested
                                    ! for any tracer
  INTEGER          :: anydrydep     ! wet deposition requested
                                    ! for any tracer
  INTEGER          :: anysedi       ! sedimentation  requested
                                    ! for any tracer
  INTEGER          :: anysemis      ! surface emission flag
                                    ! for any tracer
  INTEGER          :: anyconv       ! convection flag
  INTEGER          :: anyvdiff      ! vertical diffusion flag
  INTEGER          :: anyconvmassfix  !
  INTEGER          :: nadvec        ! number of advected tracers
  LOGICAL          :: oldrestart    ! true to read old restart
     format
  !
  ! individual information for each tracer
  !
  TYPE (t_trinfo) :: ti  (jptrac) ! Individual settings
                                  !for each tracer
  !
  ! reference to memory buffer info
  !
  TYPE (t_p_mi)   :: mi  (jptrac) ! memory buffer information
                                  !for each tracer
  TYPE (memory_info), POINTER :: mixt    ! memory buffer
                                         ! information for XT
  TYPE (memory_info), POINTER :: mixtm1  ! memory buffer
                                         ! information for XTM1
END TYPE t_trlist
```

The component ntrac gives the total number of tracers handled by the model. The components any... are derived by a bitwise OR of the corresponding individual tracer flags. Individual flags are stored in component ti of type t_trinfo. They reflect the values of the arguments passed to subroutine new_tracer.

Listing 3.27: t_trinfo

```
TYPE t_trinfo
```

```
!
! identification of transported quantity
!
CHARACTER(len=ln) :: basename    ! name (instead of xt..)
CHARACTER(len=ln) :: subname     ! optional for
                                 !'colored' tracer
CHARACTER(len=ln) :: fullname    ! name_subname
CHARACTER(len=ln) :: modulename  ! name of requesting
                                 ! sub-model
CHARACTER(len=ln) :: units       ! units
CHARACTER(len=ll) :: longname    ! long name
CHARACTER(len=ll) :: standardname   ! CF standard name
INTEGER           :: trtype  ! type of tracer:
                             ! 0=undef., 1=prescribed,
                             ! 2=diagnostic (no transport),
                             ! 3=prognostic (transported)
INTEGER           :: spid    ! species id (index in
                             ! speclist) where physical/chem.
                             ! properties are defined
INTEGER           :: nphase  ! phase (1=GAS, 2=AEROSOLMASS,
                             ! 3=AEROSOLNUMBER,...)
INTEGER           :: mode    ! aerosol mode or bin number
REAL(dp)          :: moleweight ! molecular mass (copied
                       ! from species upon initialisation)
! Requested resources ...
!
INTEGER    :: burdenid    ! index in burden diagnostics
!
! Requested resources ...
!
INTEGER  :: nbudget     ! calculate budgets (default 0)
INTEGER  :: ntran       ! perform transport (default 1)
INTEGER  :: nfixtyp     ! type of mass fixer (default 1)
INTEGER  :: nconvmassfix ! use xt_conv_massfix in cumastr
INTEGER  :: nvdiff      ! vertical diffusion flag
                        ! (default 1)
INTEGER  :: nconv       ! convection flag (default 1)
INTEGER  :: ndrydep     ! dry deposition flag:
                        ! 0=no drydep,
                        ! 1=prescribed vd,
                        ! 2=Ganzeveld
INTEGER  :: nwetdep     ! wet deposition flag (default 0)
INTEGER  :: nsedi       ! sedimentation flag (default 0)
REAL     :: tdecay      ! decay time (exponential)
                        ! (default 0.sec)
INTEGER  :: nemis   ! surface emission flag (default 0)
!
! initialization and restart
```

```
  !
  INTEGER  :: ninit  ! initialization request flag
  INTEGER  :: nrerun ! rerun flag
  REAL     :: vini   ! initialization value (default 0.)
  INTEGER  :: init   ! initialisation method actually used
  !
  ! Flags used for postprocessing
  !
  INTEGER  :: nwrite   ! write flag (default 1)
  INTEGER  :: code     ! tracer code (default 235...)
  INTEGER  :: table    ! tracer code table (default 0)
  INTEGER  :: gribbits ! bits for encoding (default 16)
  INTEGER  :: nint     ! integration (accumulation)
                       ! flag  (default 1)
  !
  ! Flags to be used by chemistry or tracer modules
  !
  INTEGER          :: nsoluble   ! soluble flag (default 0)
  TYPE(t_flag)     :: myflag (nf)! user defined flag
  type(time_days) :: tupdatel   ! last update time
  type(time_days) :: tupdaten   ! next update time
  !
END TYPE t_trinfo
```

The data type t_flag is defined as follows:

Listing 3.28: data type t_flag

```
  TYPE t_flag
    CHARACTER(len=lf) :: c       ! character string
    REAL              :: v       ! value
  END TYPE t_flag
```

The lengths of the character string components are:

Listing 3.29: Length of strings

```
  INTEGER, PARAMETER :: ln =  24 ! length of name
                                 ! (char) components
  INTEGER, PARAMETER :: ll = 256 ! length of
                                 ! longname component
  INTEGER, PARAMETER :: lf =   8 ! length of flag
                                 ! character string
  INTEGER, PARAMETER :: nf =  10 ! number of user
                                 ! defined flags
  INTEGER, PARAMETER :: ns =  20 ! max number of submodels
```