PALM Code Structure & Features

Institute of Meteorology and Climatology, Leibniz Universität Hannover

PALM Code Structure & Features

General remarks

- This lecture gives a brief overview about the code structure of PALM.
- Please note:
  There is ongoing work on further modularization of the PALM code, which will affect parts of the program structure that is presented in this lecture.

Overview

- PALM is written in FORTRAN2008.
- With some very minor exceptions, the code is using the FORTRAN standard, so it should compile without error on any FORTRAN 2003/2008 compiler (90/95 may give problems).
- Machine dependent code segments, e.g. calls of routines from external libraries (e.g. NetCDF or FFTW), which may not be available on some machines, are activated using preprocessor directives.
- The serial and parallel (MPI) PALM version is also activated by preprocessor directives.
- The automatic installer automatically sets the approriate preprocessor directives. For manual settings of directives see https://palm.muk.uni-hannover.de/trac/wiki/doc/app/cpp_options.

PALM Code Structure & Features

Overview

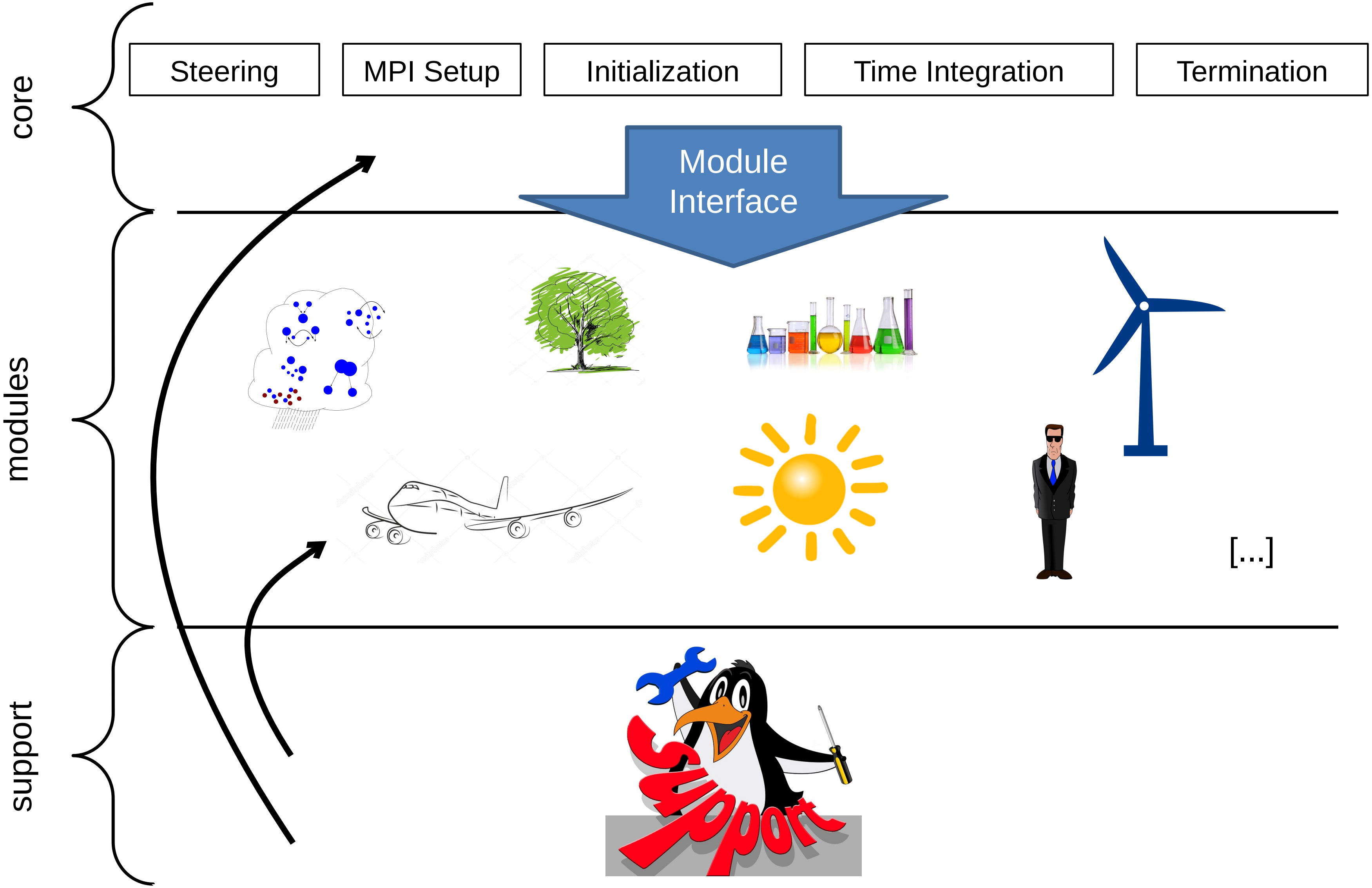
- The code is divided into several files, each file containing:
  - a single MODULE (ending with _mod.f90), including several associated SUBROUTINEs, or
  - a single SUBROUTINE, e.g. file parin.f90 contains SUBROUTINE parin.
- PALM includes a special user module (user_module.f90) designed to add additional code written by the user.

Why should you use the user module instead of directly modiying the source code?

- The user module very rarely changes in future PALM releases and can be easily re-used by newer versions of PALM without requiring extensive changes.

PALM Code Structure & Features

General structure



core

Steering

MPI Setup

Initialization

Time Integration

Termination

Module Interface

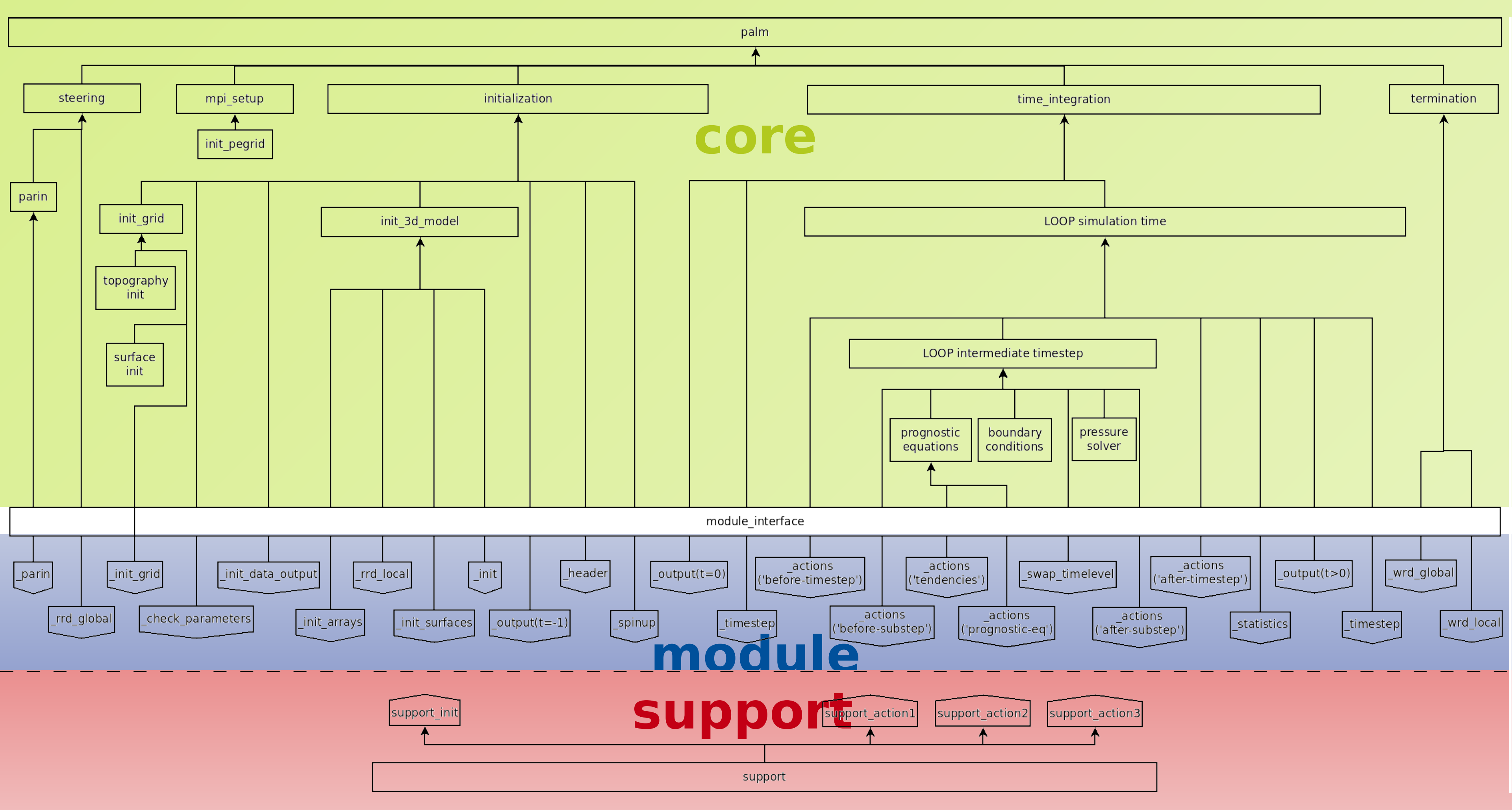
modules

[...]

support

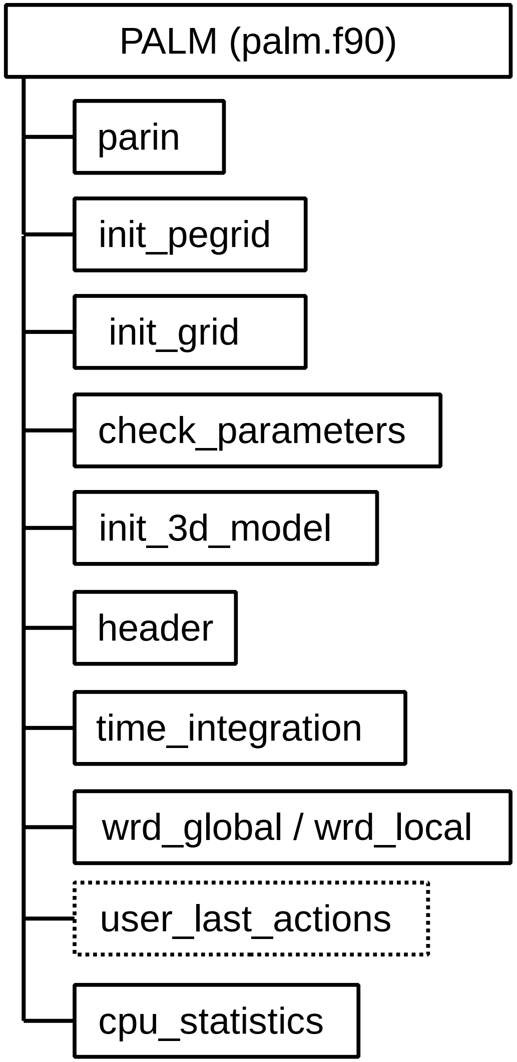
PALM Code Structure & Features

Code structure



PALM Code Structure & Features

Detailed structure



PALM (palm.f90)

- parin
- init_pegrid
- init_grid
- check_parameters
- init_3d_model
- header
- time_integration
- wrd_global / wrd_local
- user_last_actions
- cpu_statistics

▪ Read steering parameters

▪ Create processor topology (for parallel PALM mode)

▪ Create grid-depending constants

▪ Check setting & dependencies of steering parameters

▪ Allocate arrays and initialize 3D model

▪ Output important information about current run

▪ Integrate model equations (statistical analysis, data output)

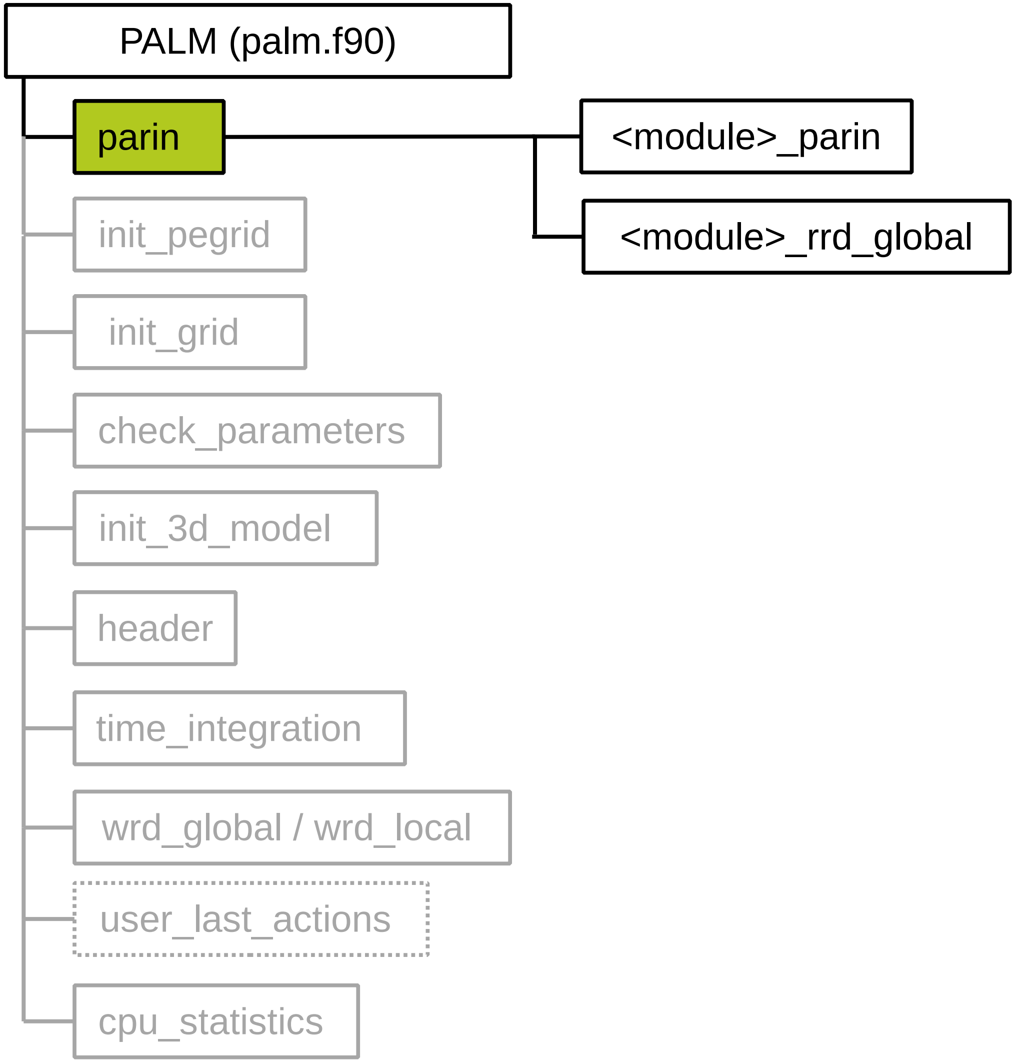
▪ Binary output for restarts

▪ Execute user-defined last actions (if user module is used)

▪ Output of measured cpu times

PALM Code Structure & Features

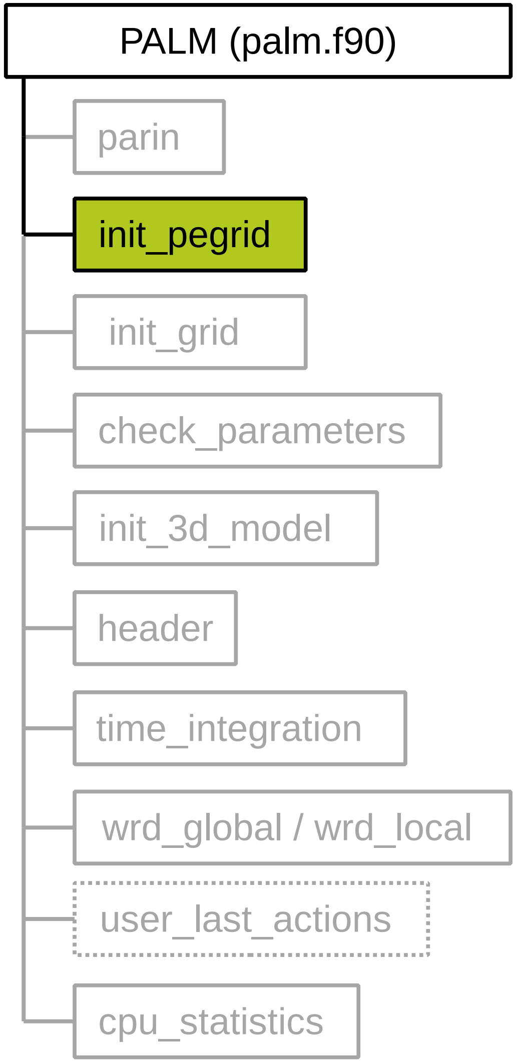
Detailed structure



- Read input parameters from namelist file
- Read control parameters from restart file in case of restart run

PALM Code Structure & Features

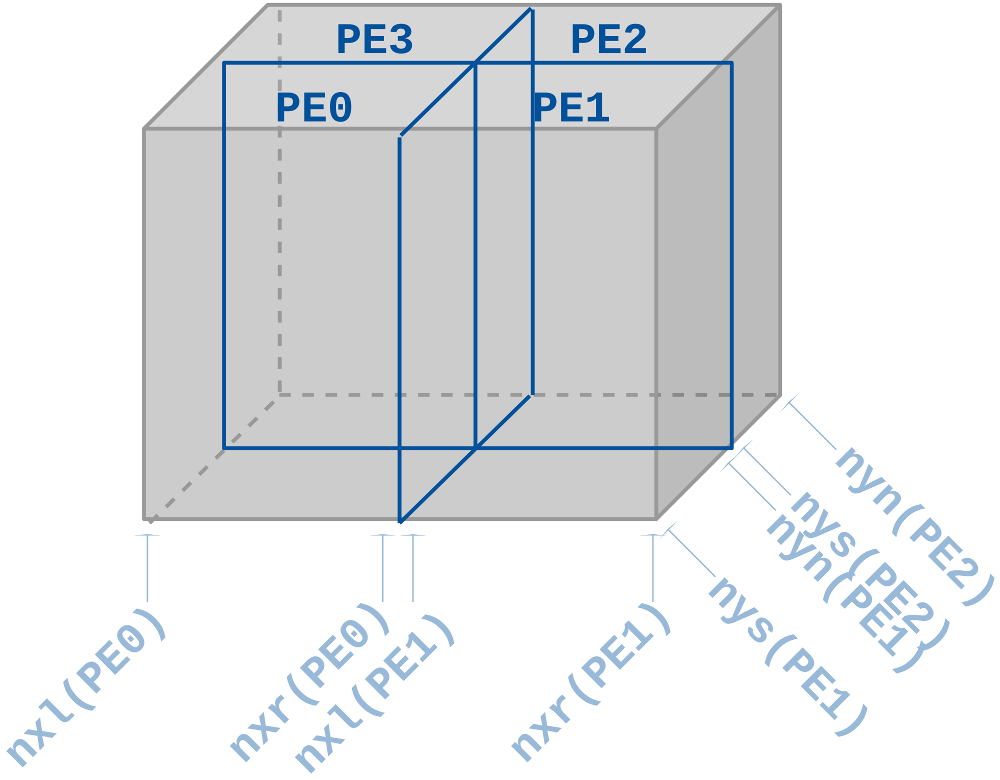
Detailed structure



PALM (palm.f90)

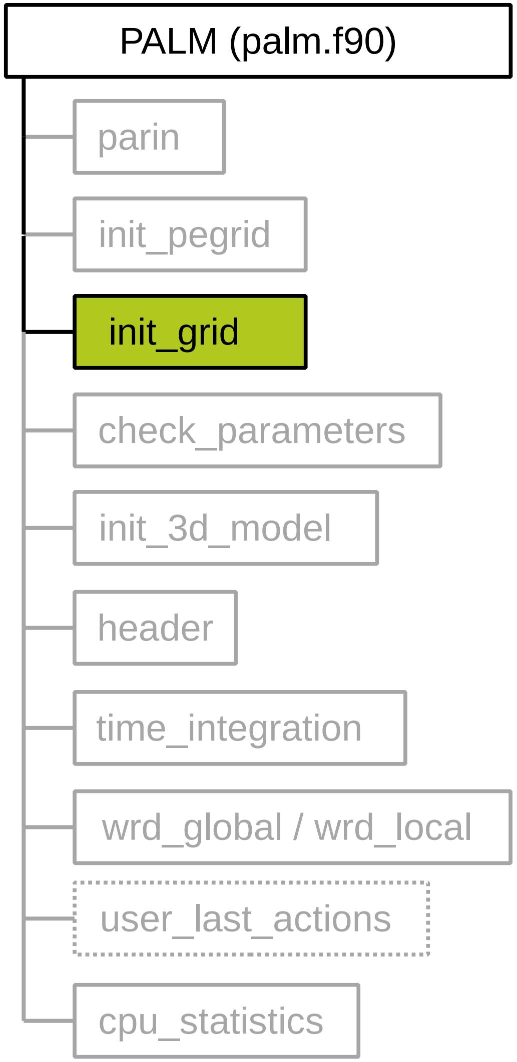
- parin
- init_pegrid
- init_grid
- check_parameters
- init_3d_model
- header
- time_integration
- wrd_global / wrd_local
- user_last_actions
- cpu_statistics

- Determination of virtual processor topology (if not prescribed by user) & computation of grid point number and array bounds of local subdomains



PALM Code Structure & Features

Detailed structure



PALM (palm.f90)

parin

init_pegrid

init_grid

check_parameters

init_3d_model

header

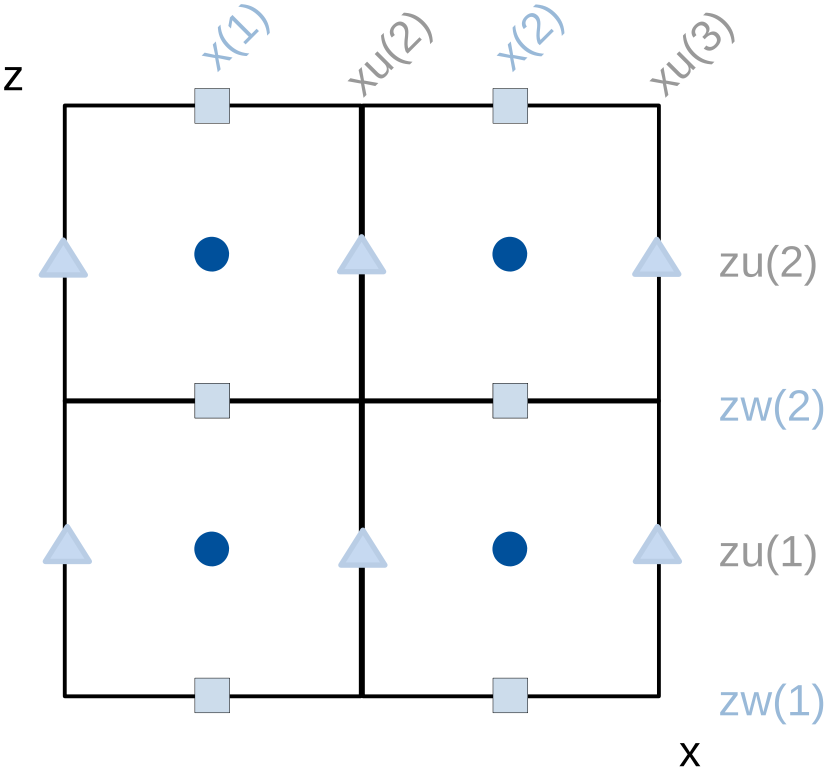
time_integration

wrd_global / wrd_local

user_last_actions

cpu_statistics

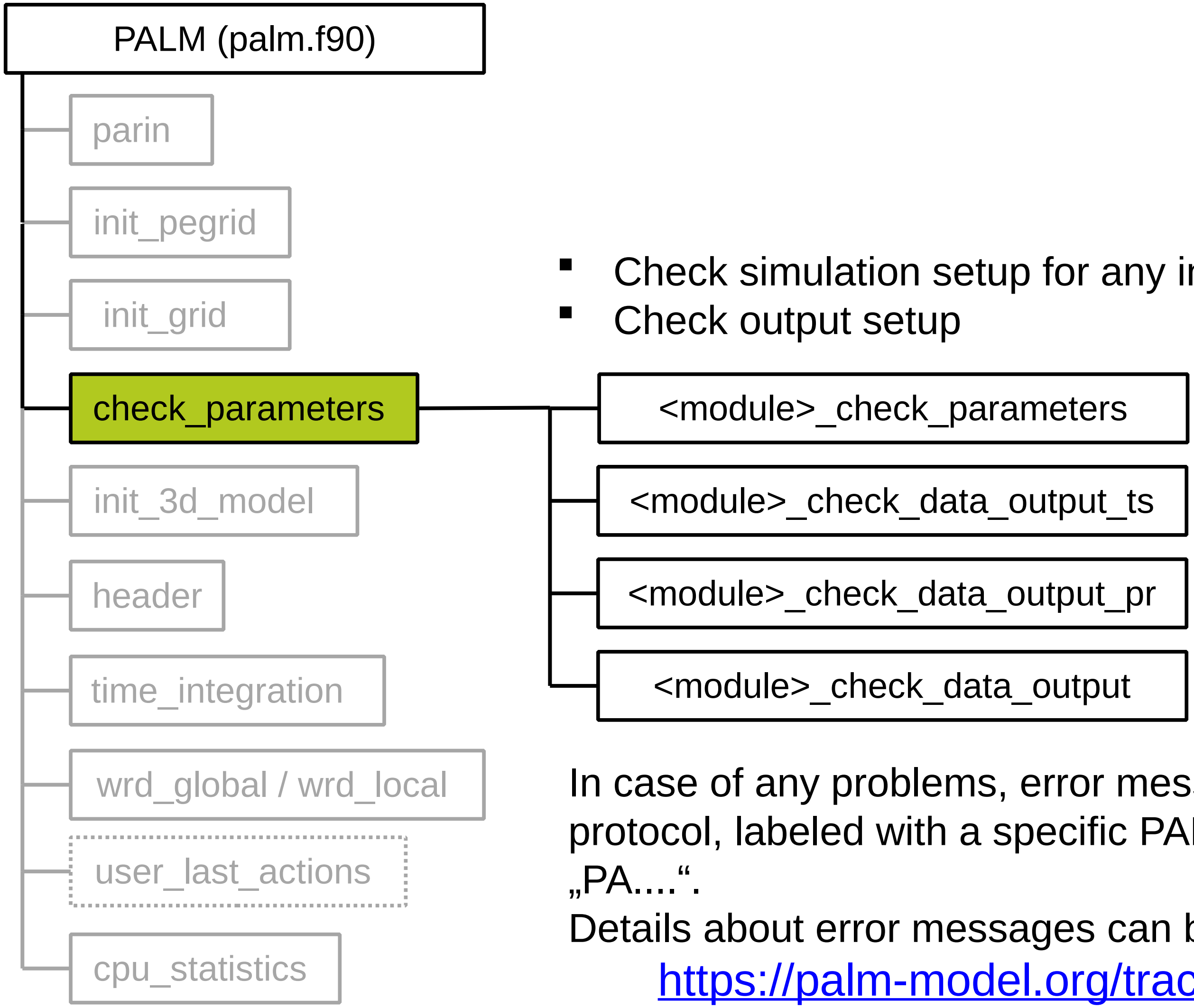
- Pre-calculation of metric grid coordinates on staggered PALM grid, e.g.:



PALM Code Structure & Features

Detailed structure

- Check simulation setup for any inconsistencies
- Check output setup



PALM (palm.f90)

parin

init_pegrid

init_grid

check_parameters

init_3d_model

header

time_integration

wrd_global / wrd_local

user_last_actions

cpu_statistics

<module>_check_parameters

<module>_check_data_output_ts

<module>_check_data_output_pr

<module>_check_data_output

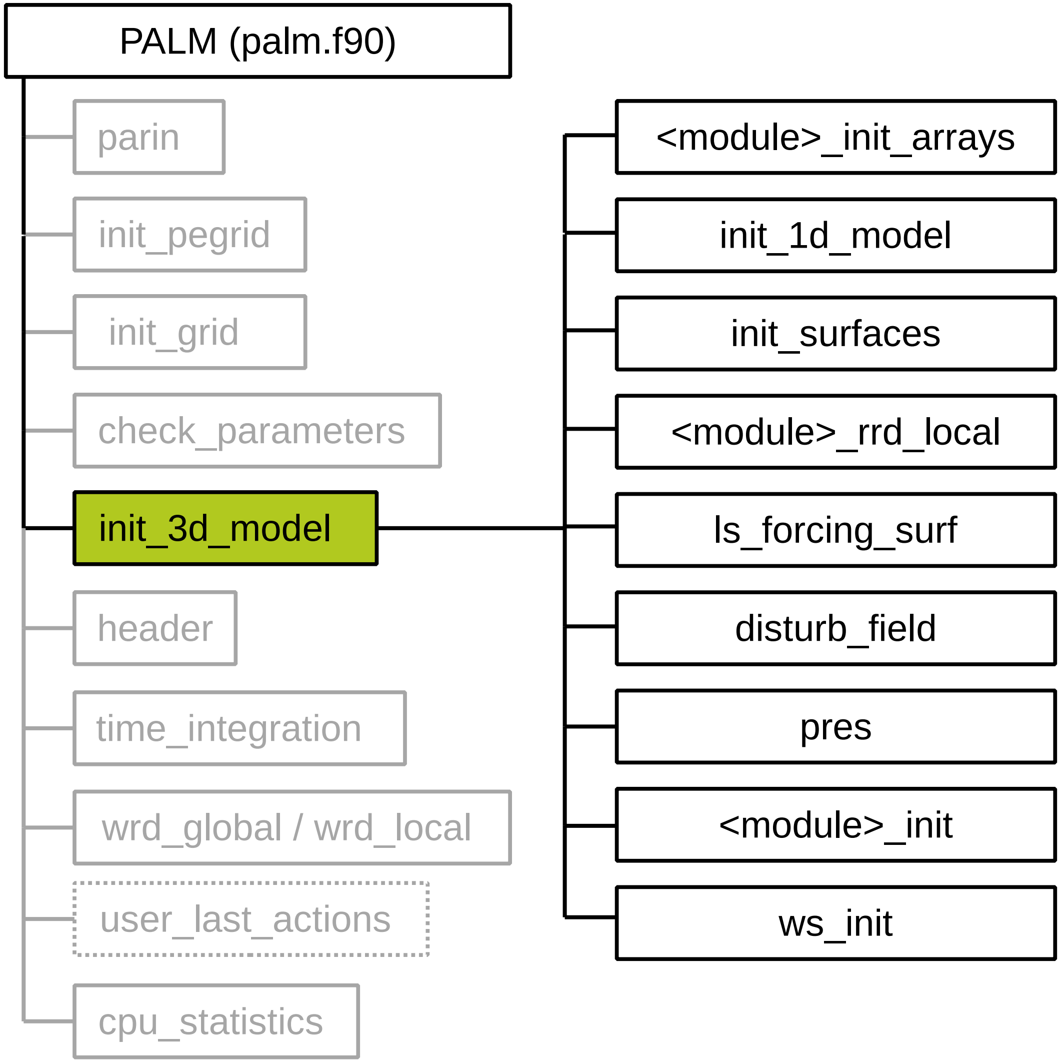
In case of any problems, error messages appear in the job protocol, labeled with a specific PALM error code number „PA....“.

Details about error messages can be found at:

https://palm-model.org/trac/wiki/doc/app/errmsg

PALM Code Structure & Features

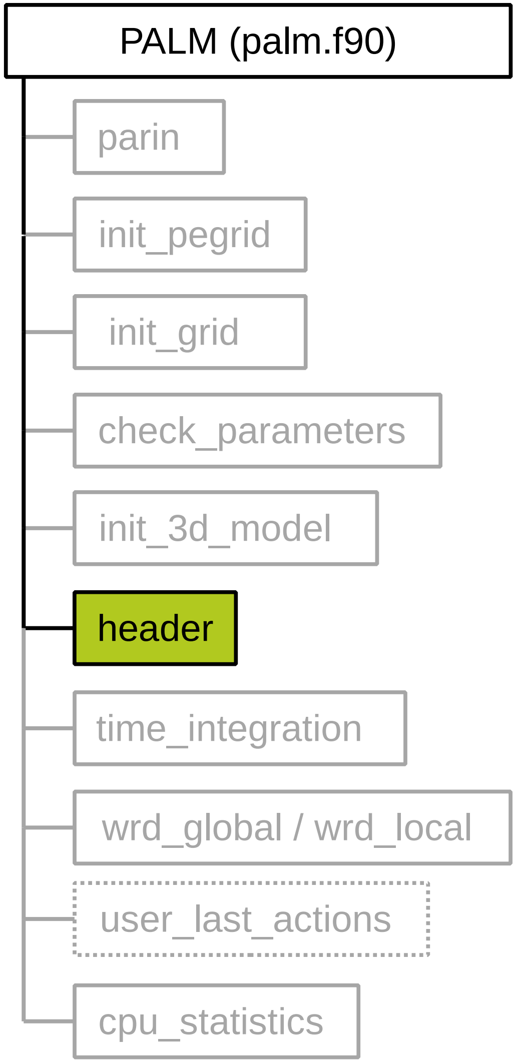
Detailed structure



- Initialize module-specific arrays
- Pre-calculate profiles via 1D model
- Initialize surfaces
- Read arrays from restart data files
- Load large-scale forcing
- Impose random perturbations
- Call pressure solver
- Initialization actions of the modules
- Initialize Wicker-Skamarock scheme

PALM Code Structure & Features

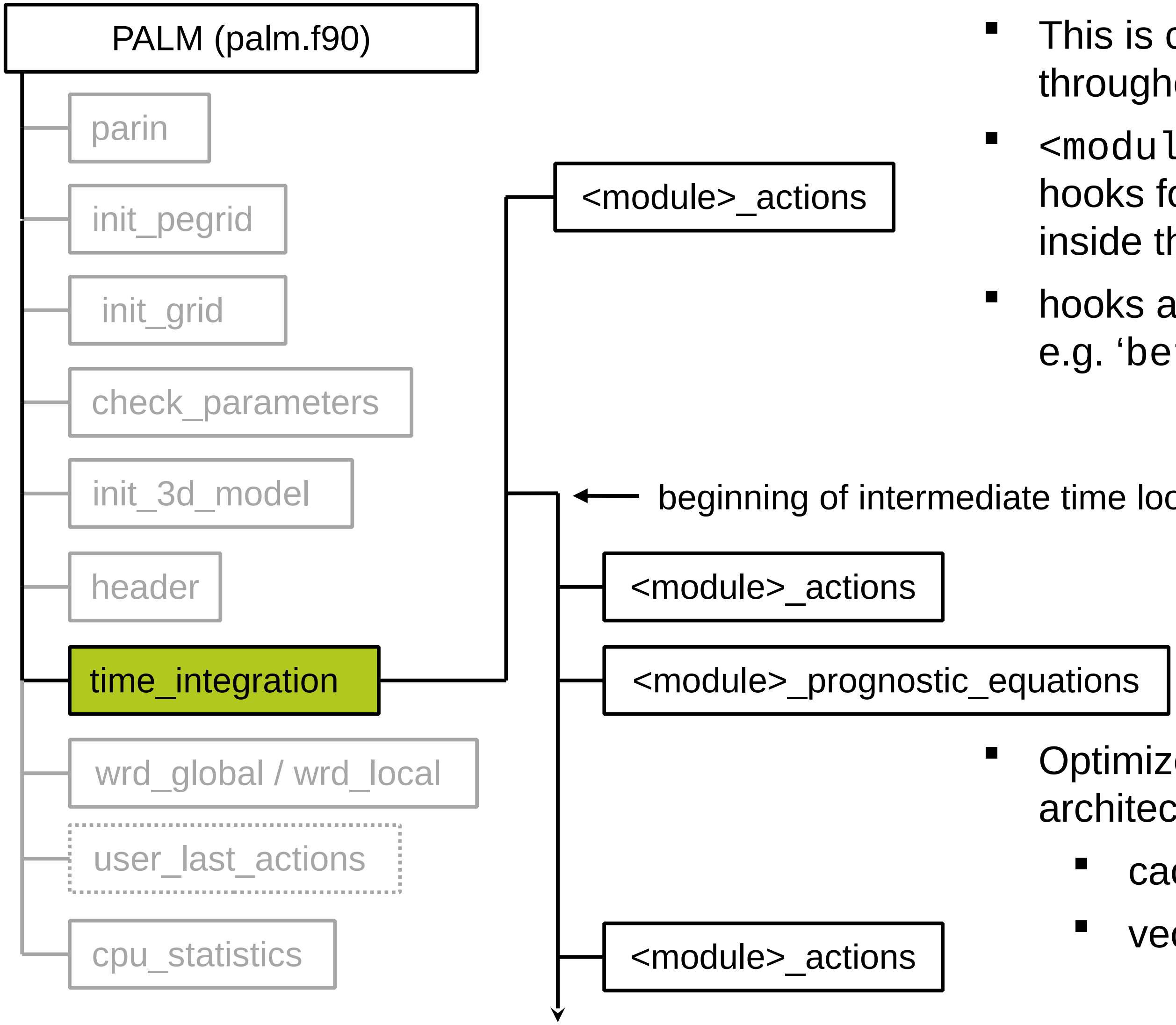
Detailed structure



- Write information about steering parameters to file, useful for job monitoring (see lecture “PALM steering”)

PALM Code Structure & Features

Detailed structure

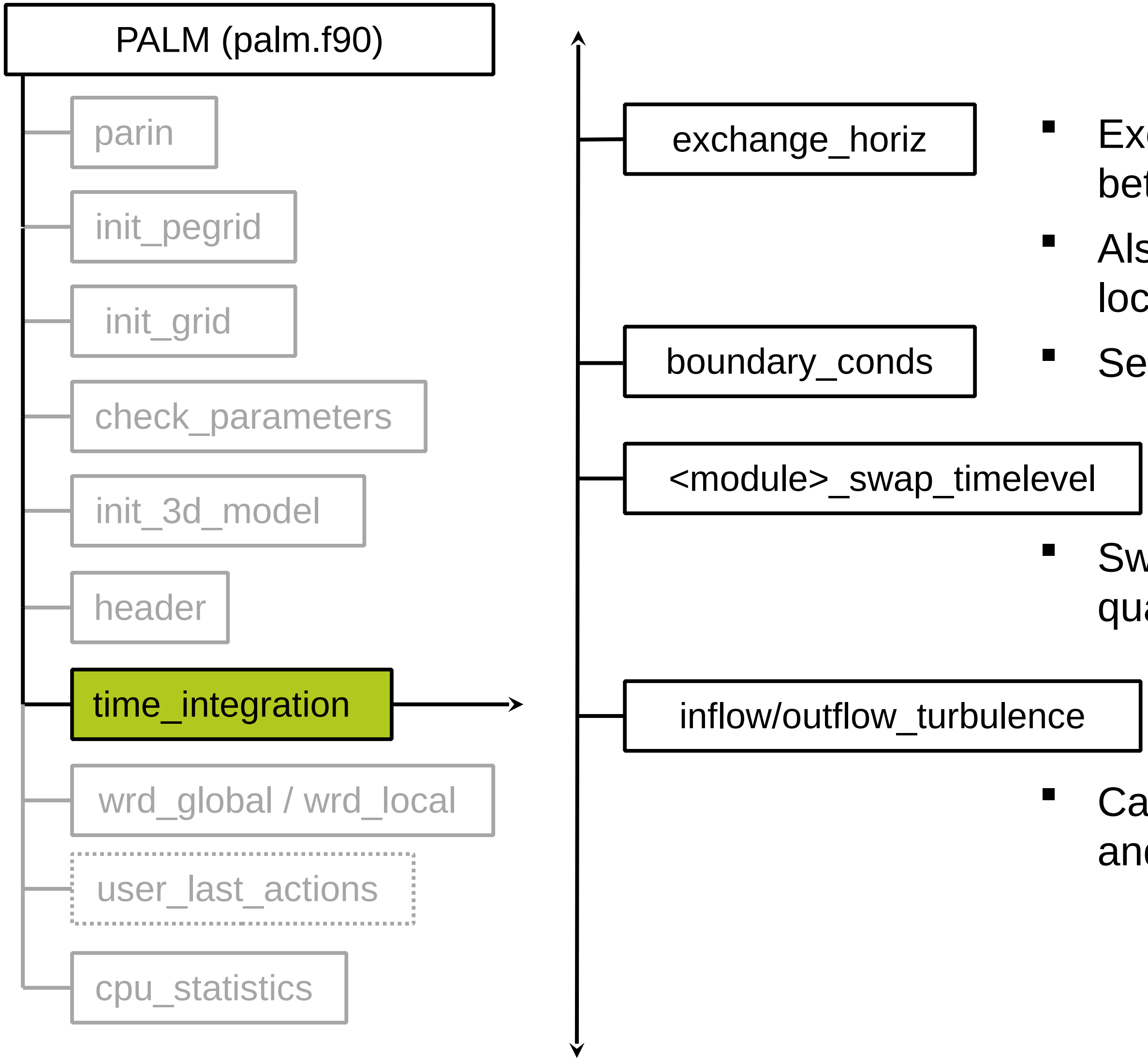


- This is called at several positions throughout `time_integration`
- `<module>_action` has several hooks for different positions inside the time loop
- hooks are addressed via strings, e.g. ‘`before_timestep`’

- Optimized for different computer architectures
  - cache-based machines
  - vector-based machines

PALM Code Structure & Features

Detailed structure



PALM (palm.f90)

- parin
- init_pegrid
- init_grid
- check_parameters
- init_3d_model
- header
- time_integration
- wrd_global / wrd_local
- user_last_actions
- cpu_statistics

exchange_horiz

- Exchange of ghost-point data between neighbouring PEs
- Also called at some other locations

boundary_conds

- Setting of boundary conditions

<module>_swap_timelevel

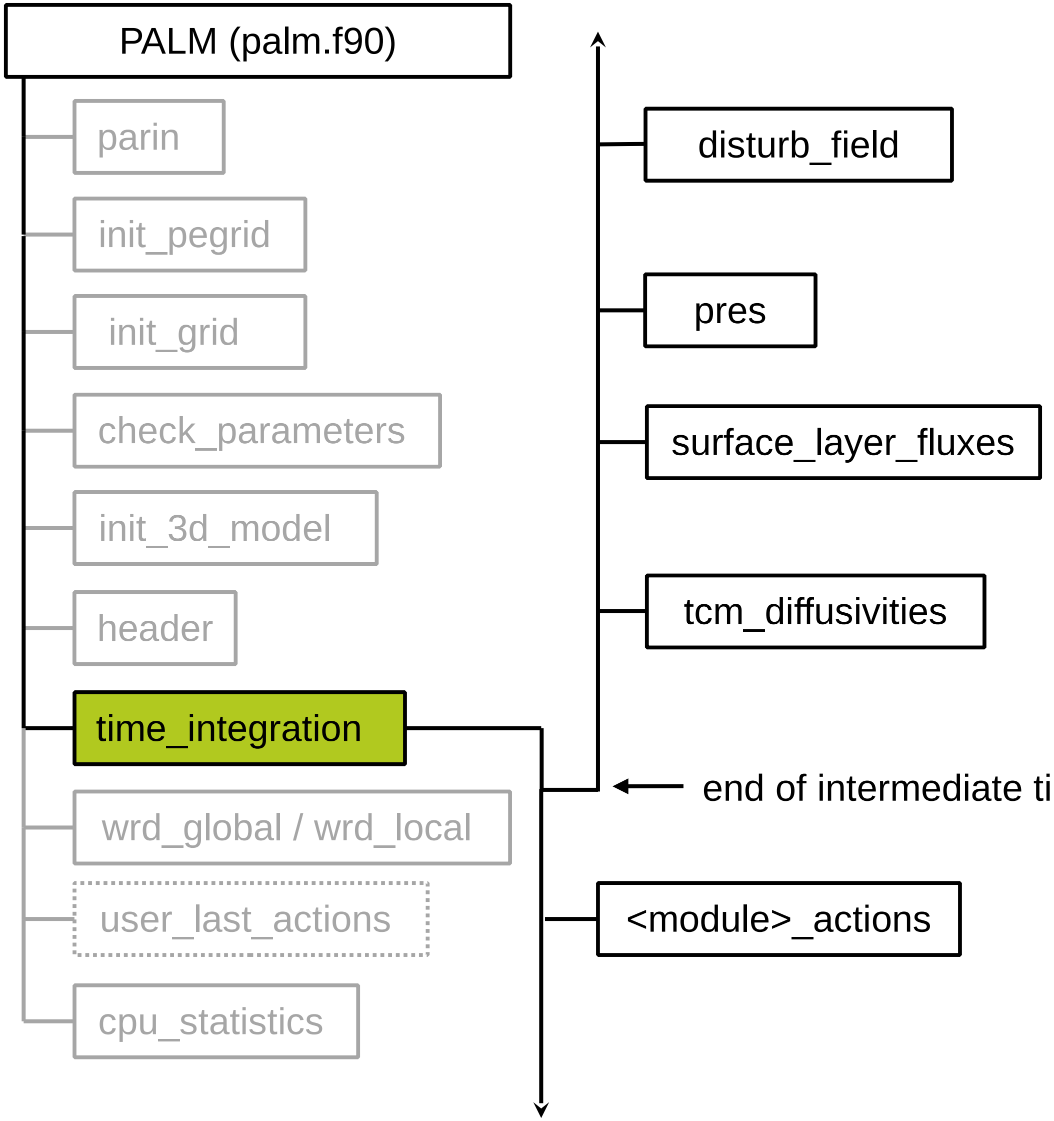
- Swapping of prognostic quantities data

inflow/outflow_turbulence

- Calculating turbulent inflow and/or outflow conditions

PALM Code Structure & Features

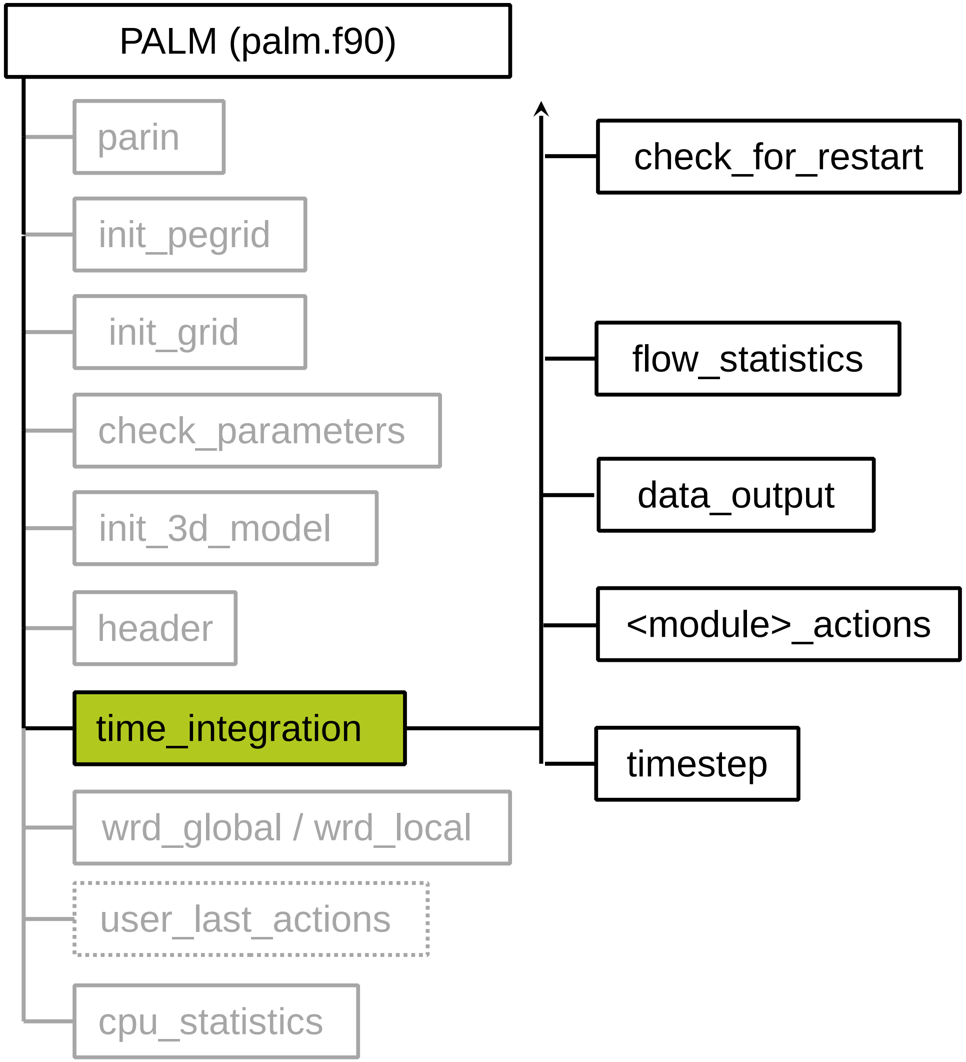
Detailed structure



- Random perturbations are imposed to horizontal velocity components
- Pressure solver
- Vertical turbulent fluxes in the surface (constant-flux) layer are computed
- Computing of diffusion coefficients

PALM Code Structure & Features

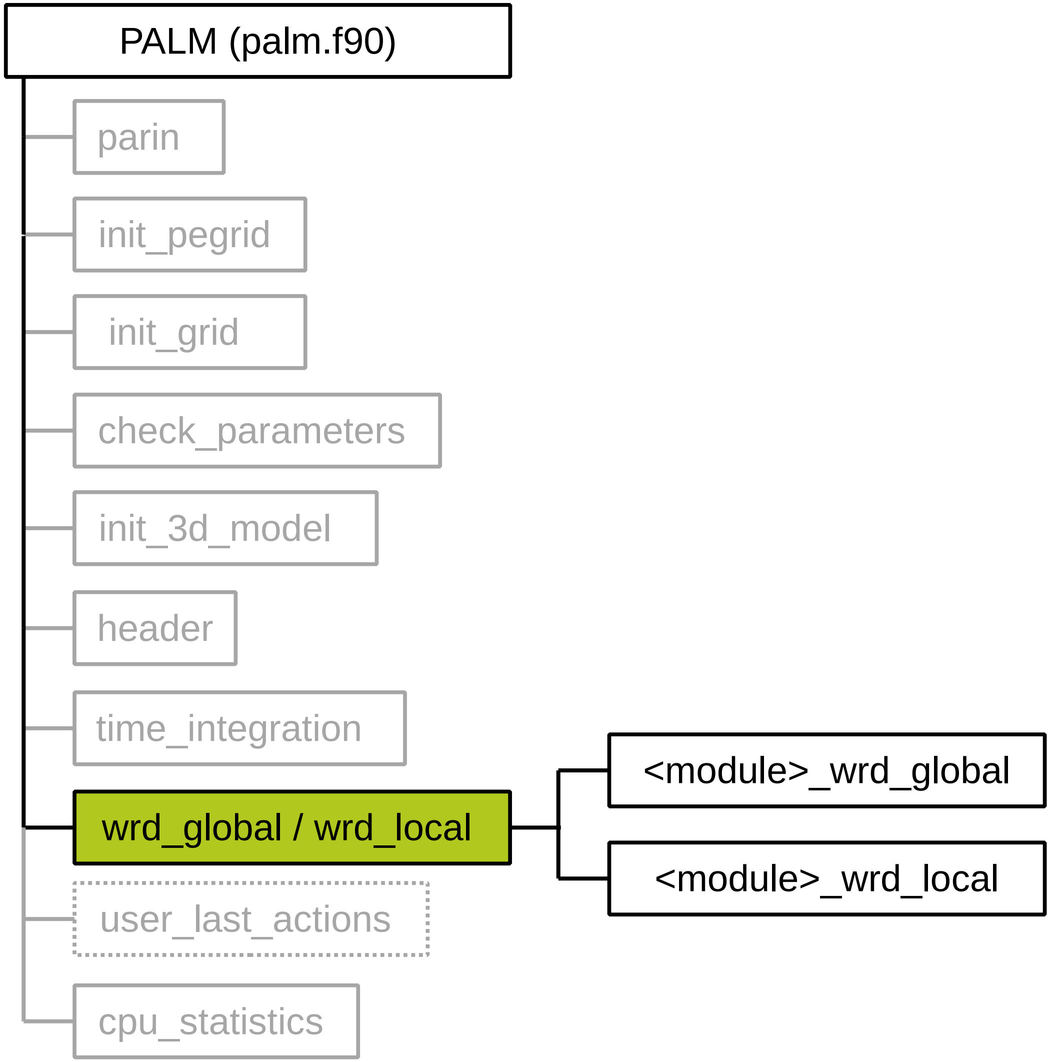
Detailed structure



- Checking if run needs to be terminated (due to insufficient remaining CPU time) and prepare for restart
- Flow statistics are calculated
- Output of requested variables
- Calculating next timestep width

PALM Code Structure & Features

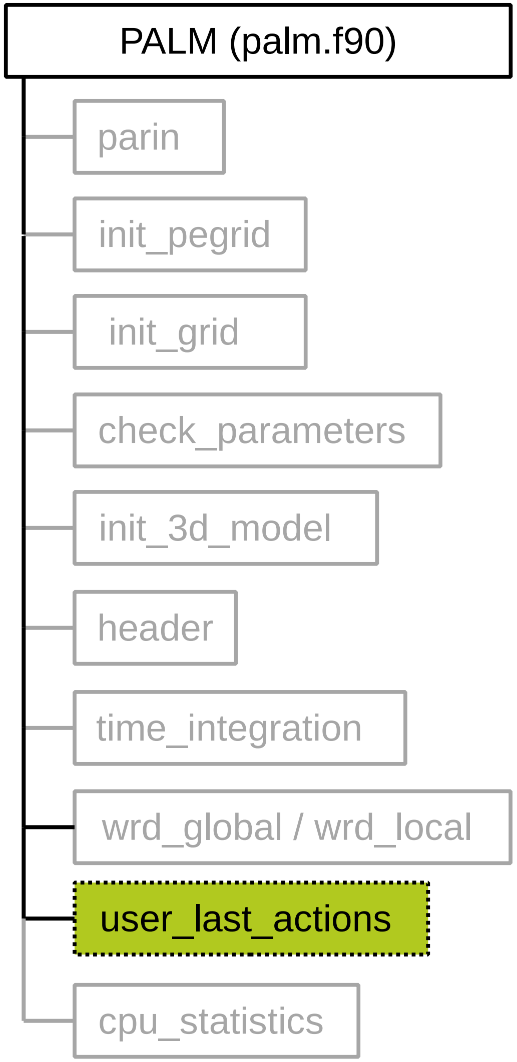
Detailed structure



- Write data required for restarts
  - Steering parameters
  - 3d arrays of prognostic variables

PALM Code Structure & Features

Detailed structure



PALM (palm.f90)

parin

init_pegrid

init_grid

check_parameters

init_3d_model

header

time_integration

wrd_global / wrd_local

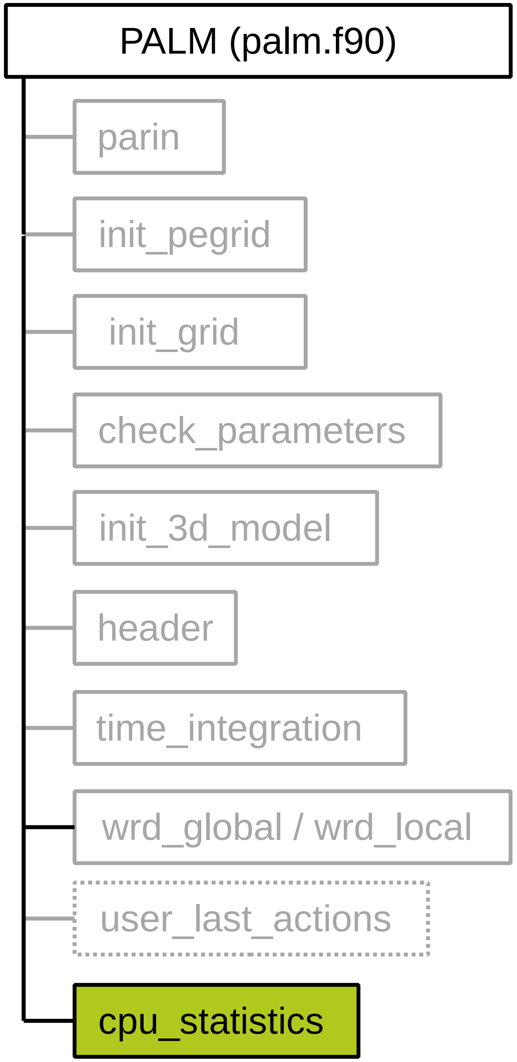
user_last_actions

cpu_statistics

- Any other last actions before end of simulation

PALM Code Structure & Features

Detailed structure



PALM (palm.f90)

parin

init_pegrid

init_grid

check_parameters

init_3d_model

header

time_integration

wrd_global / wrd_local

user_last_actions

cpu_statistics

- Calculate and output information about required CPU time, in total and for certain parts of PALM, e.g. “all progn. equations” or “pres” (see lecture “PALM steering”)

Content – Part 2

- Global variables
- Preprocessor directives
- Automatic documentation using Doxygen
- Important variables and their declaration

PALM Code Structure & Features

Global variables

- Global variables and parameters are defined in **`modules.f90`**
- Only variables that are used in multiple parts of PALM
- Defined in different **`MODULE`**s, e.g., **`pt`** is defined in **`arrays_3d`**:

```
MODULE arrays_3d
[...]
    REAL(wp), DIMENSION(:,:,:), ALLOCATABLE, TARGET :: pt
[...]
END MODULE
```

PALM Code Structure & Features

Global variables

- **USE** statement to load variables in other routines:

```
SUBROUTINE buoyancy( var, wind_component )

    USE arrays_3d,                                &
        ONLY:  pt, pt_slope_ref, ref_state, tend

    USE control_parameters,                       &
        ONLY:  atmos_ocean_sign, cos_alpha_surface
[...]

END SUBROUTINE buoyancy
```

PALM Code Structure & Features

Preprocessor directives

- Preprocessor directives are special lines in the code which allow to compile alternative parts of the code depending on so-called “**define string** switches“
- Code example:

```
#if defined ( __parallel )
    CALL MPI_ALLREDUCE( a, b, nzt-nzb, MPI_REAL, MPI_SUM,
                        comm2d, ierr )
#else
    b = a
#endif
```

- If the compiler is called as

  ```
      ifort -fpp -D__parallel ...
  ```

  then the **#if** branch is compiled

- If the compiler is called **without** option –D__parallel
  the **#else** branch is compiled

PALM Code Structure & Features

Preprocessor directives

- The preprocessor directives require to activate their processing with a specific compiler option, which is e.g. `-cpp` for the Intel compiler.
- Preprocessor directives and options have to be given in the **`%cpp_options`** line of the palmrun configuration file **`(.palm.config.<ci>`**, see lecture “PALM steering“)
- Preprocessor options may differ for different compilers
- Define-string switches can be combined using logical AND (**&&**) / OR (**||**) operators:

```
#if defined( __abc && __def )
#if defined( __abc || __def )
```

- Logical NOT operator:

```
#if ! defined( __abc )
```

PALM Code Structure & Features

Preprocessor directives

Additional use of preprocessor directives

- Replacing strings in the code, e.g.,

```
%cpp_options -cpp -DMPI_REAL=MPI_DOUBLE_PRECISION
```

replaces **MPI_REAL** with **MPI_DOUBLE_PRECISION** **before** compiling

- Planned use: switch ON/OFF entire modules, e.g. Lagrangian particle model
  - Unnecessary code is not compiled
  - Less compiling time
  - Less memory consumption

PALM Code Structure & Features

List of define-string switches used in PALM

PALM mode	__parallel	Parallel PALM version
	__single_precision	Use 32-bit arithmetic (still in test phase)
System specific	__ibm	IBM Regatta systems
	__nec	NEC-SX systems
Software specific	__intel_compiler	Compilers
	__mpifh	Old MPI libraries
	__netcdf, __netcdf4, __netcdf4_parallel	NetCDF I/O with different NetCDF versions
	__fftw	Fast FFT
	__rrtmg	Radiative transfer model
	__rrtmg	External radiation model library (see lecture)

- Switches set under `%cpp_options` in `.palm.config.<ci>` file are automatically used by `palmbuild` for compiling.

PALM Code Structure & Features

Automatic documentation using Doxygen

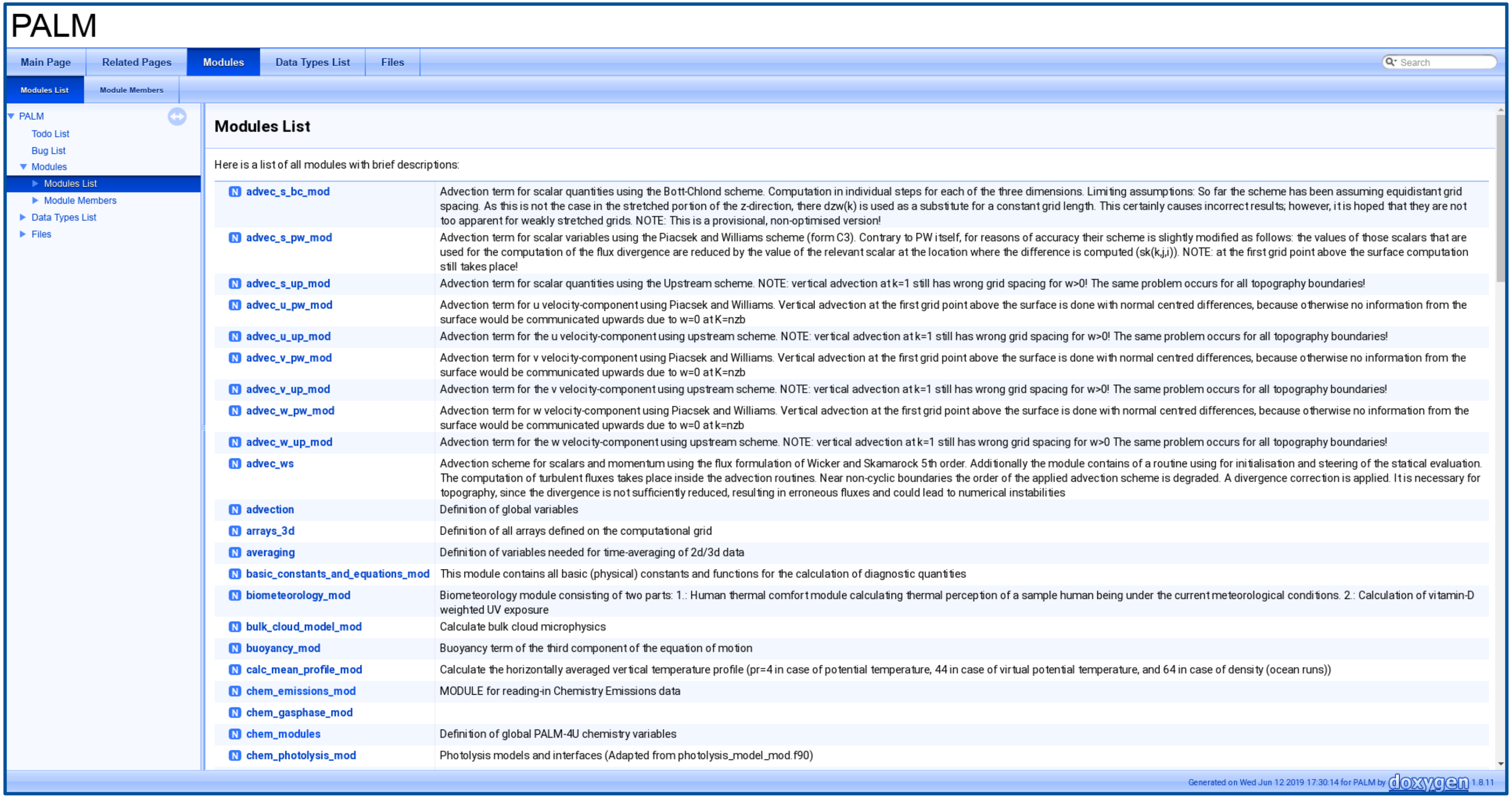
- Doxygen:
  “Doxygen is the de facto standard tool for generating documentation from annotated C++ sources, but it also supports other popular programming languages such as [...] ***Fortran*** *[...].“*
- Automatic documentation from the source code via tags.
- **Tags** currently used in PALM:

Description of variables	`REAL :: ol !< Obukhov length`
File/Routine description	`!> This Routine does things`
To do lists	`!> @todo Missing implementation of...`
Bugs	`!> @bug 1D model crashes when...`

Important notes	`!> @note Soil layer must not be`
	`!>       too thin`

PALM Code Structure & Features

Doxygen demonstration



Modules List

Here is a list of all modules with brief descriptions:

N advec_s_bc_mod	Advection term for scalar quantities using the Bott-Chlond scheme. Computation in individual steps for each of the three dimensions. Limiting assumptions: So far the scheme has been assuming equidistant grid spacing. As this is not the case in the stretched portion of the z-direction, there dzw(k) is used as a substitute for a constant grid length. This certainly causes incorrect results; however, it is hoped that they are not too apparent for weakly stretched grids. NOTE: This is a provisional, non-optimised version!
N advec_s_pw_mod	Advection term for scalar variables using the Piacsek and Williams scheme (form C3). Contrary to PW itself, for reasons of accuracy their scheme is slightly modified as follows: the values of those scalars that are used for the computation of the flux divergence are reduced by the value of the relevant scalar at the location where the difference is computed (sk(k,j,i)). NOTE: at the first grid point above the surface computation still takes place!
N advec_s_up_mod	Advection term for scalar quantities using the Upstream scheme. NOTE: vertical advection at k=1 still has wrong grid spacing for w>0! The same problem occurs for all topography boundaries!
N advec_u_pw_mod	Advection term for u velocity-component using Piacsek and Williams. Vertical advection at the first grid point above the surface is done with normal centred differences, because otherwise no information from the surface would be communicated upwards due to w=0 at K=nzb
N advec_u_up_mod	Advection term for the u velocity-component using upstream scheme. NOTE: vertical advection at k=1 still has wrong grid spacing for w>0! The same problem occurs for all topography boundaries!
N advec_v_pw_mod	Advection term for v velocity-component using Piacsek and Williams. Vertical advection at the first grid point above the surface is done with normal centred differences, because otherwise no information from the surface would be communicated upwards due to w=0 at K=nzb
N advec_v_up_mod	Advection term for the v velocity-component using upstream scheme. NOTE: vertical advection at k=1 still has wrong grid spacing for w>0! The same problem occurs for all topography boundaries!
N advec_w_pw_mod	Advection term for w velocity-component using Piacsek and Williams. Vertical advection at the first grid point above the surface is done with normal centred differences, because otherwise no information from the surface would be communicated upwards due to w=0 at k=nzb
N advec_w_up_mod	Advection term for the w velocity-component using upstream scheme. NOTE: vertical advection at k=1 still has wrong grid spacing for w>0 The same problem occurs for all topography boundaries!
N advec_ws	Advection scheme for scalars and momentum using the flux formulation of Wicker and Skamarock 5th order. Additionally the module contains of a routine using for initialisation and steering of the statical evaluation. The computation of turbulent fluxes takes place inside the advection routines. Near non-cyclic boundaries the order of the applied advection scheme is degraded. A divergence correction is applied. It is necessary for topography, since the divergence is not sufficiently reduced, resulting in erroneous fluxes and could lead to numerical instabilities
N advection	Definition of global variables
N arrays_3d	Definition of all arrays defined on the computational grid
N averaging	Definition of variables needed for time-averaging of 2d/3d data
N basic_constants_and_equations_mod	This module contains all basic (physical) constants and functions for the calculation of diagnostic quantities
N biometeorology_mod	Biometeorology module consisting of two parts: 1.: Human thermal comfort module calculating thermal perception of a sample human being under the current meteorological conditions. 2.: Calculation of vitamin-D weighted UV exposure
N bulk_cloud_model_mod	Calculate bulk cloud microphysics
N buoyancy_mod	Buoyancy term of the third component of the equation of motion
N calc_mean_profile_mod	Calculate the horizontally averaged vertical temperature profile (pr=4 in case of potential temperature, 44 in case of virtual potential temperature, and 64 in case of density (ocean runs))
N chem_emissions_mod	MODULE for reading-in Chemistry Emissions data
N chem_gasphase_mod	
N chem_modules	Definition of global PALM-4U chemistry variables
N chem_photolysis_mod	Photolysis models and interfaces (Adapted from photolysis_model_mod.f90)

Generated on Wed Jun 12 2019 17:30:14 for PALM by doxygen 1.8.11

How to use Doxygen

- Install **Doxygen** and **dot** on your system.
- Run script: **`palmdocs`**
- At the end of the output, **`palmdocs`** will tell you, where to find the newly generated HTML Documentation. To view the docs, open the file **`PALM_doc.html`** in your browser.

PALM Code Structure & Features

Important variables and their declaration

- 3D-arrays of prognostic variables are named Ψ, and Ψ_p for time level t, and $t + \Delta t$, respectively, with $\Psi = u, v, w, pt, q, s, e, sa, \ldots$

- They are by default declared as $\Psi(z,y,x)$ or $\Psi(k,j,i)$, e.g.

```
u(nzb:nzt+1,nysg:nyng,nxlg:nxrg)
```

with

```
nysg = nys – nbgp,  nyng = nyn + nbgp
nxlg = nxl – nbgp,  nxrg = nxr + nbgp
nzb, nzt  (bottom/top grid index)
nys, nyn  (south/north grid index)
nxl, nxr  (left/right grid index)
```

as the index limits of the (sub-)domain.

- **nbgp** is the number of ghost points which depends on the advection scheme (**nbgp** = 3 for the default Wicker-Skamarock scheme).

PALM Code Structure & Features

Important variables and their declaration

- If only a single process/core is used, then

```
nxl = 0;  nxr = nx
nys = 0;  nyn = ny
```

- For performance optimization, most of the 3D-variables are declared as pointers, e.g.

```
REAL(wp), DIMENSION(:,:,:), POINTER ::  u, u_p
```

- This does not affect the usage of these variables in the code in (almost) any way.

Important variables and their declaration

variable	index bounds	meaning	comment

zu	**nzb:nzt+1**	heights of the scalar (u,v) grid levels	**zu(1) = 0.5*dz(1)**
			zu(0) = -zu(1)

zw	**nzb:nzt+1**	heights of the w grid levels	**zw(0) = 0**
dz	**1:10**	vertical grid spacings	to be set in **&initialization-parameters**
dzu	**1:nzt+1**	vertical grid spacings between scalar grid levels	**dzu(k) = zu(k)-zu(k-1)**
ddzu	**1:nzt+1**	inverse of grid spacings	**ddzu(k) = 1.0/dzu(k)**
dx		grid spacing along x	to be set in **&initialization-parameters**
ddx		inverse of **dx**	**ddx(k) = 1.0/dx**
current_timestep_number		timestep counter	
time_since_reference_point		simulated time in seconds	