PALM Program Structure

PALM group

Institute of Meteorology and Climatology, Leibniz Universität Hannover

last update: 21st September 2015

Contents

This part gives a brief overview about the structure of the PALM code:

Contents

This part gives a brief overview about the structure of the PALM code:

- Flow chart

Contents

This part gives a brief overview about the structure of the PALM code:

- Flow chart
- Most important variables and how they are declared

Contents

This part gives a brief overview about the structure of the PALM code:

- Flow chart
- Most important variables and how they are declared
- Machine dependencies

PALM Code: General Features

PALM Code: General Features

- PALM is written in FORTRAN90. Current version number is 3.10.

PALM Code: General Features

- PALM is written in FORTRAN90. Current version number is 3.10.
- 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)

PALM Code: General Features

- PALM is written in FORTRAN90. Current version number is 3.10.
- 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)
- Data handling between subroutines is mostly done using FORTRAN90-modules instead of using parameter lists.

```
SUBROUTINE parin                              SUBROUTINE parin( a, b, c, ... )
   USE control_parameters, ONLY: ...             INTEGER(iwp) ::  a, b
   USE grid_variables, ONLY: ...                 REAL(wp)     ::  c, ...
   .                                             .
   .                                             .
```

Most modules can be found in file .../trunk/SOURCE/modules.f90

PALM Code: General Features

PALM Code: General Features

- 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.

PALM Code: General Features

- 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.

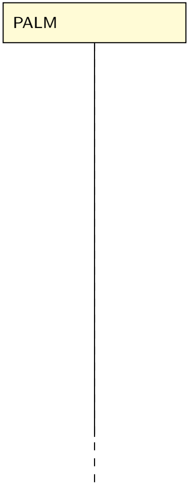
PALM Code: General Features

- 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 code is splitted into several files, most of them containing just one subroutine, e.g. file “parin.f90” contains “SUBROUTINE parin”.

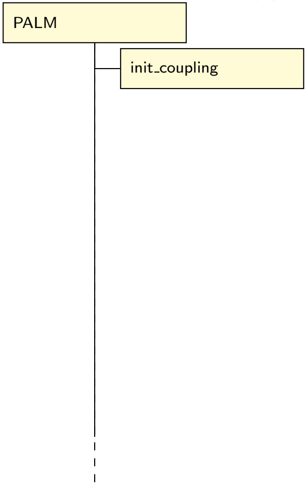
PALM Code: General Features

- 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 code is splitted into several files, most of them containing just one subroutine, e.g. file "parin.f90" contains "SUBROUTINE parin".
- The code includes an interface which can be used to add your own code extensions. Advantage: These code extensions can be reused (normally) for future PALM releases without requiring any changes.

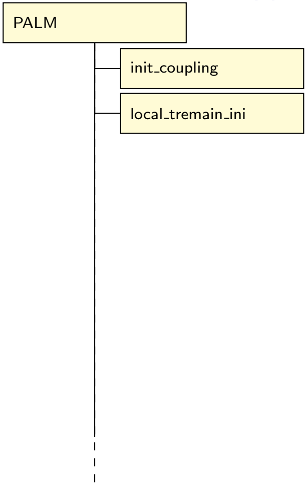
PALM Flow Chart (I)



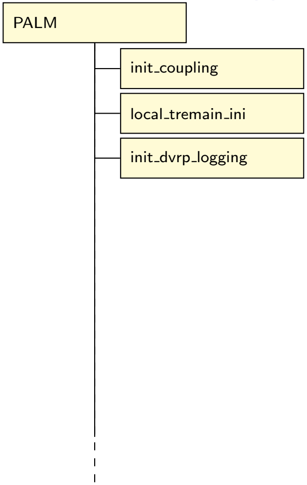
PALM Flow Chart (I)



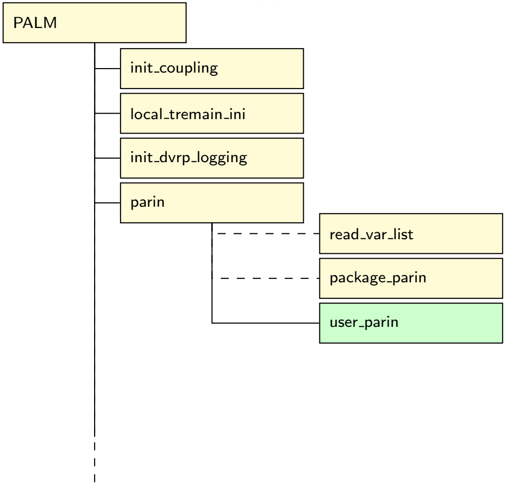
PALM Flow Chart (I)



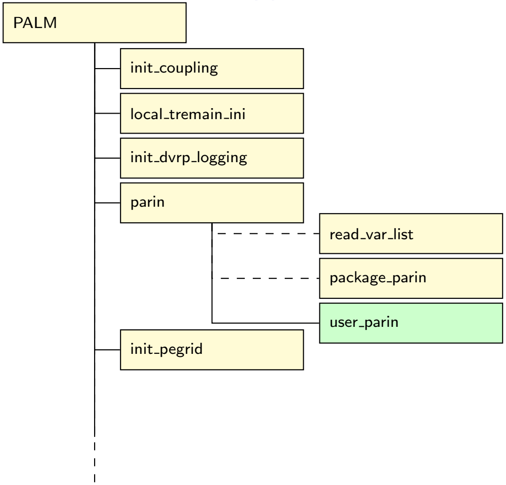
PALM Flow Chart (I)



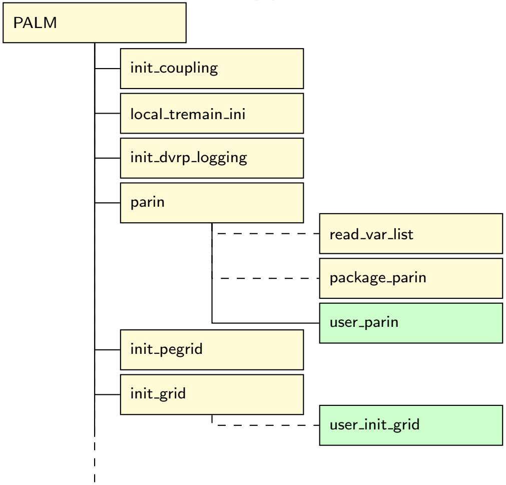
PALM Flow Chart (I)



PALM Flow Chart (I)



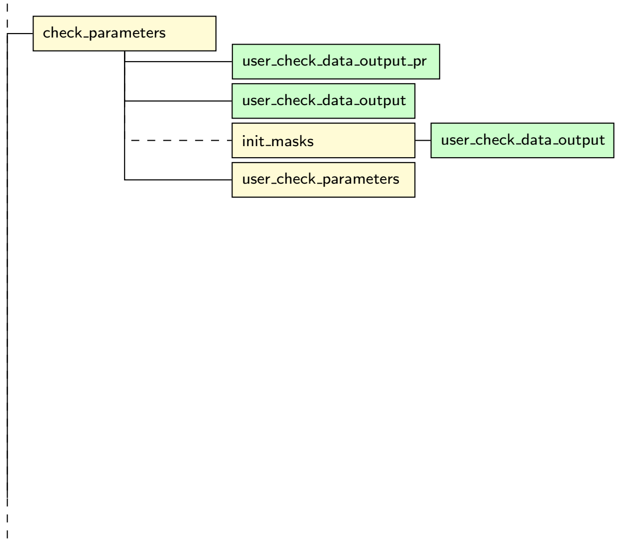
PALM Flow Chart (I)



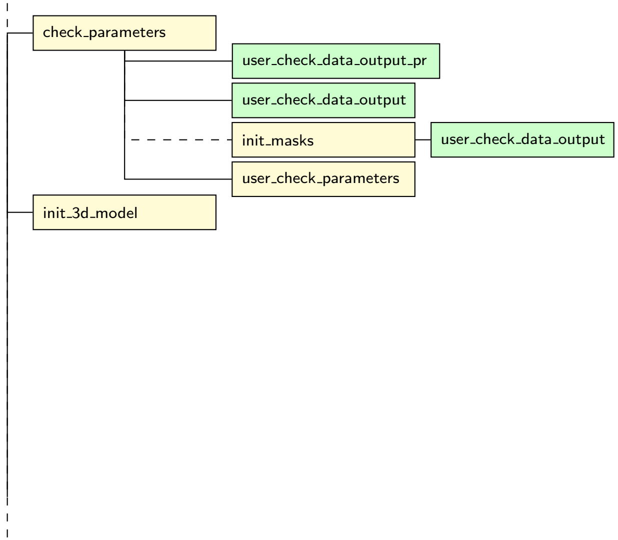
PALM Flow Chart (II)

check_parameters

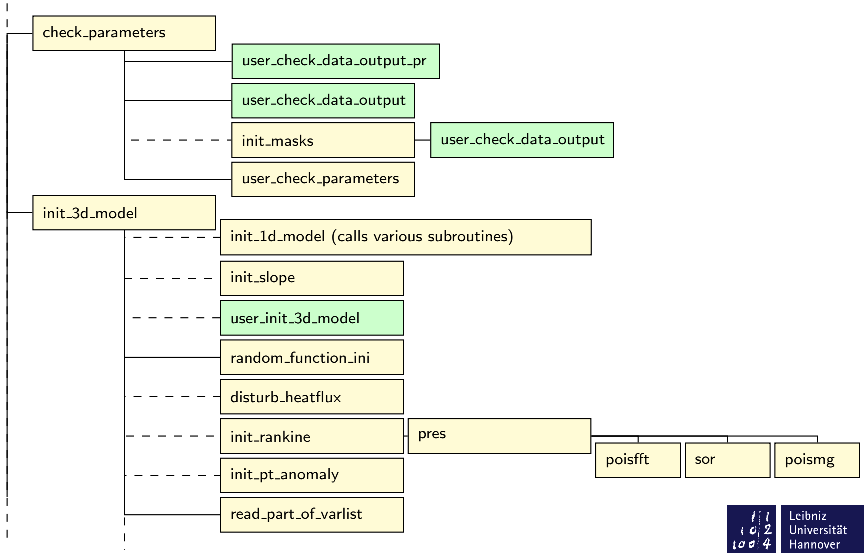
PALM Flow Chart (II)



PALM Flow Chart (II)



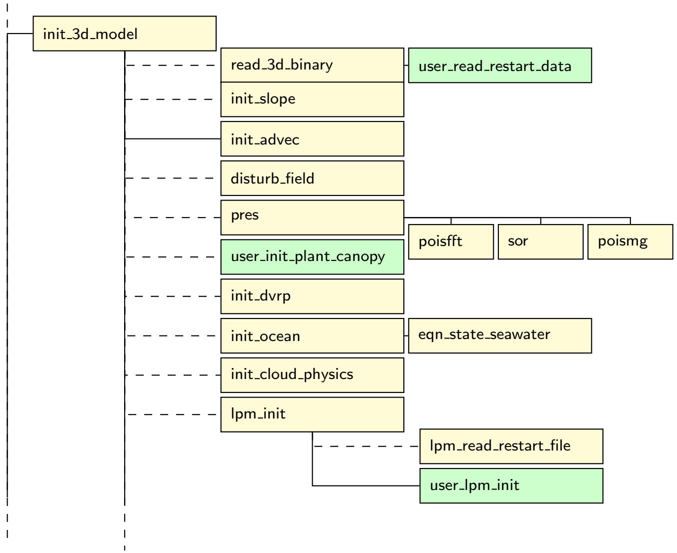
PALM Flow Chart (II)



PALM Flow Chart (III)

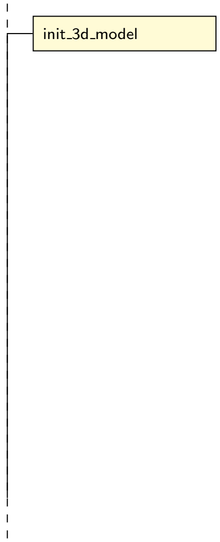
init_3d_model

PALM Flow Chart (III)

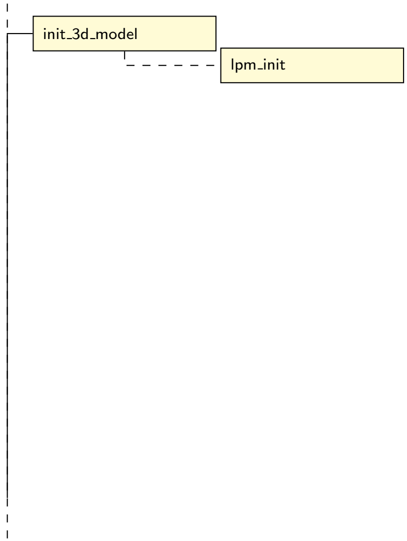


PALM Flow Chart (IV)

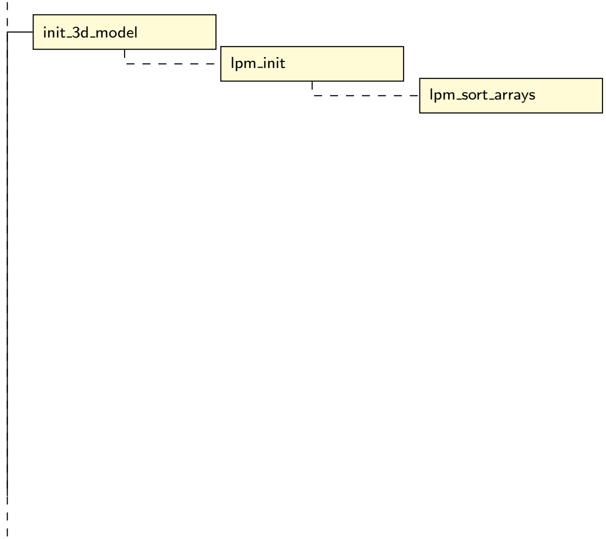
PALM Flow Chart (IV)



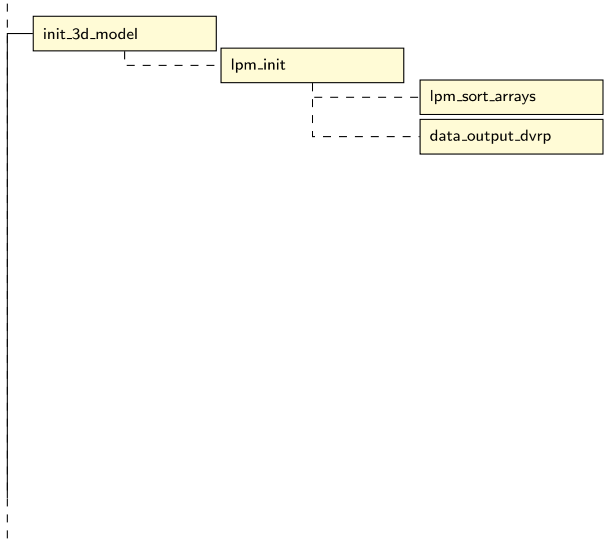
PALM Flow Chart (IV)



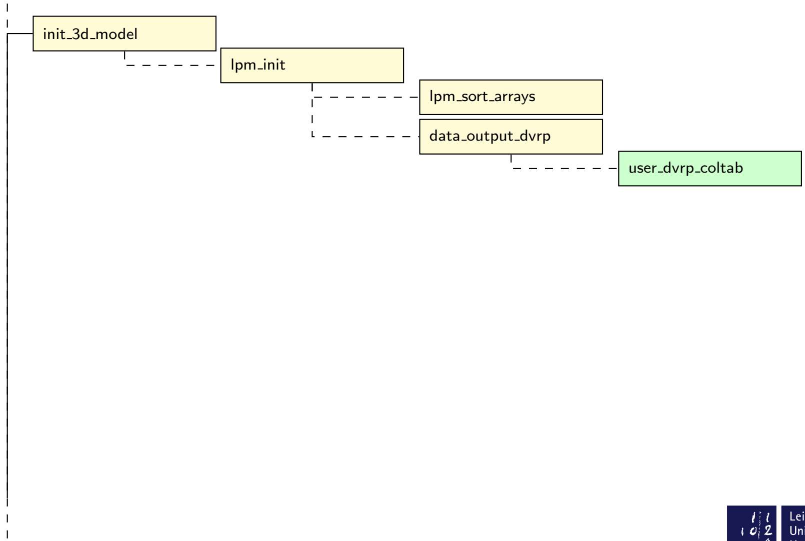
PALM Flow Chart (IV)



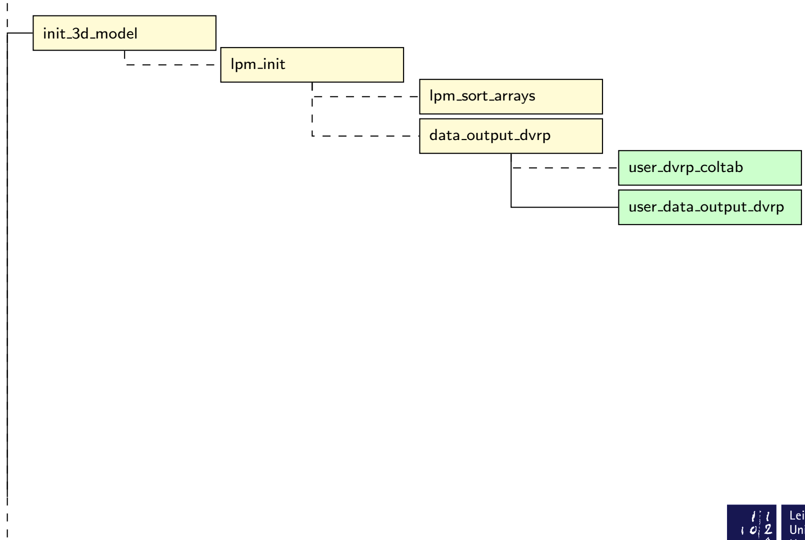
PALM Flow Chart (IV)



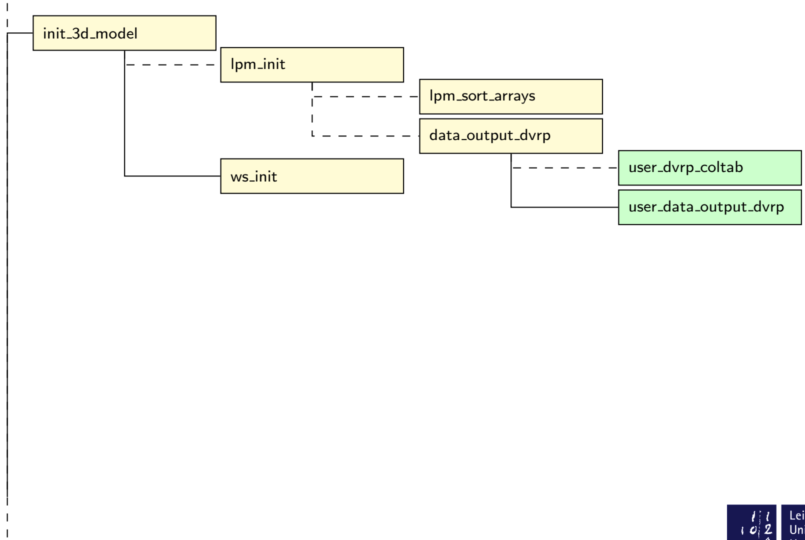
PALM Flow Chart (IV)



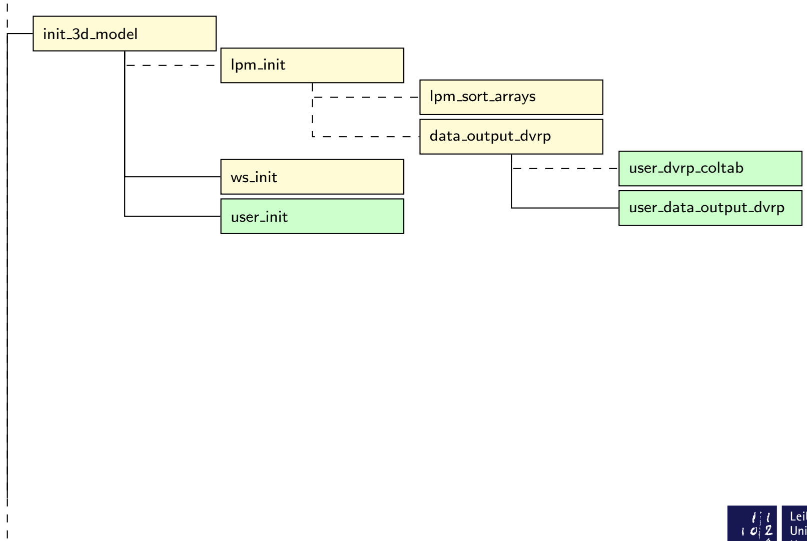
PALM Flow Chart (IV)



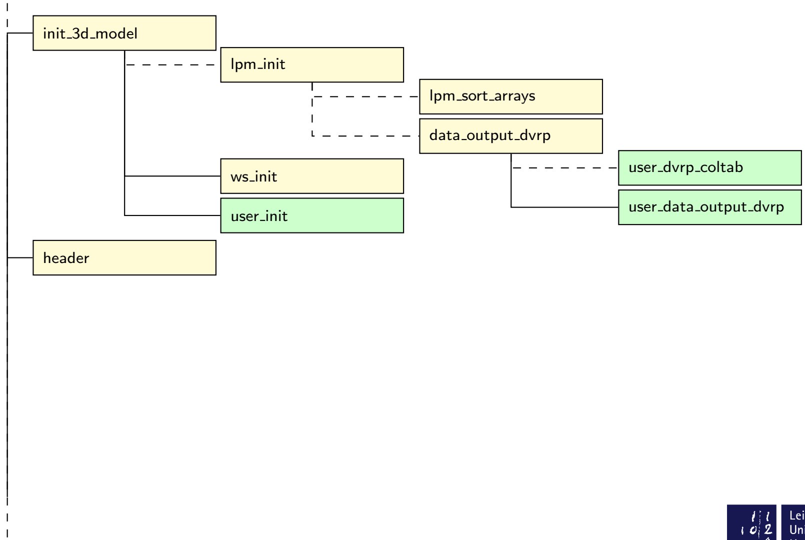
PALM Flow Chart (IV)



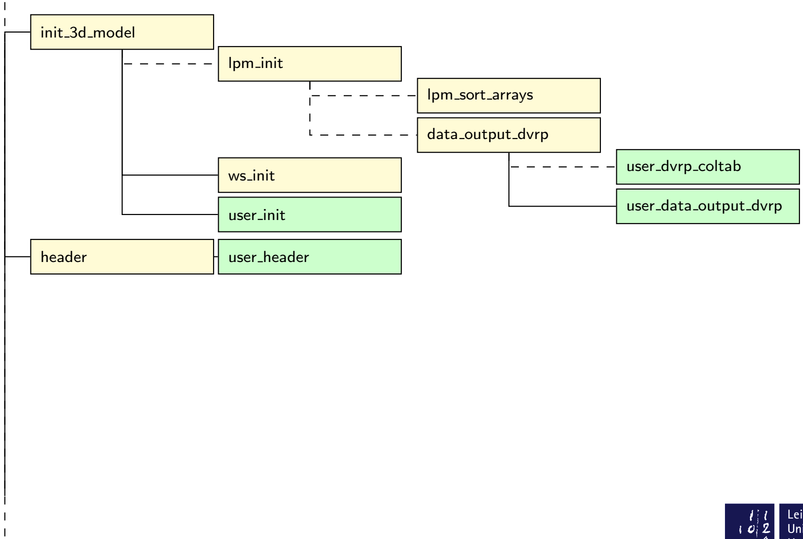
PALM Flow Chart (IV)



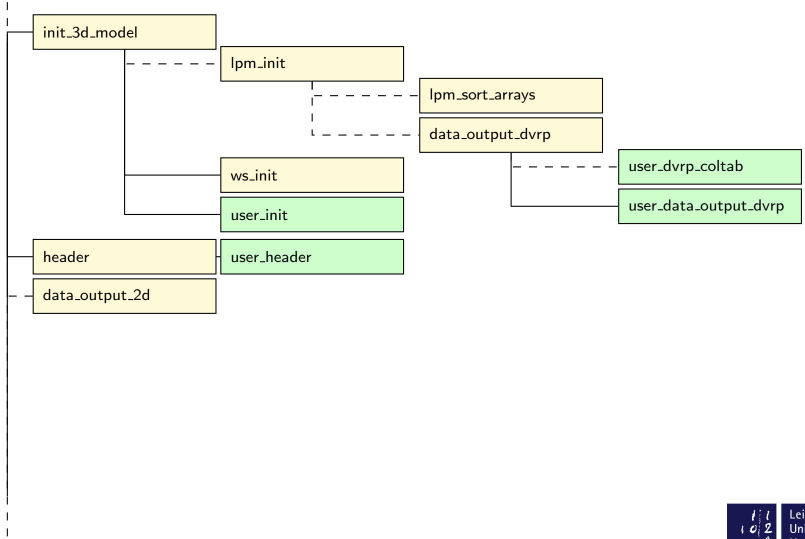
PALM Flow Chart (IV)



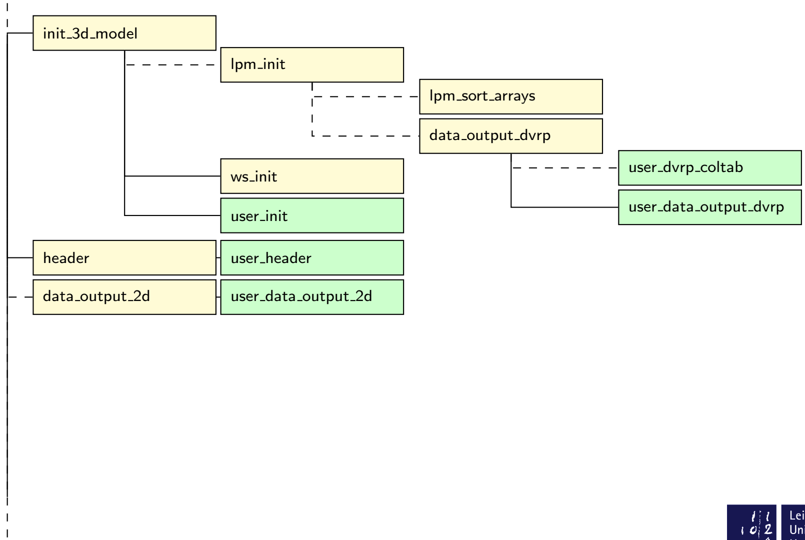
PALM Flow Chart (IV)



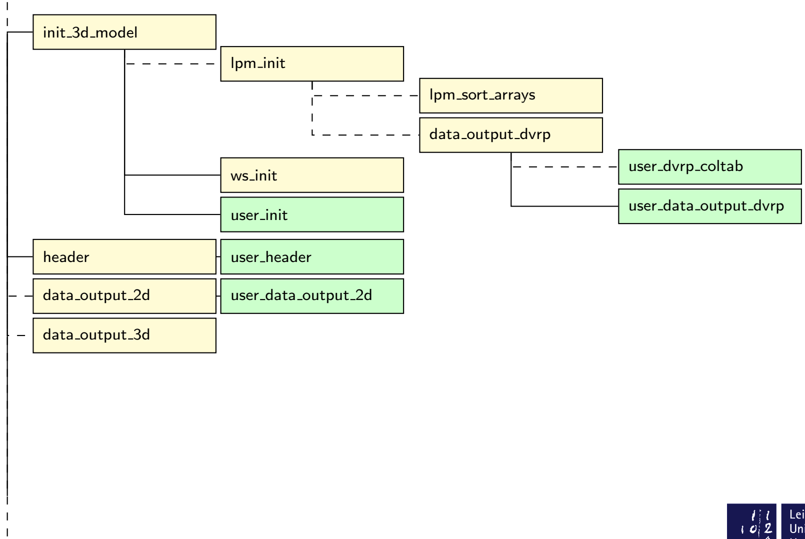
PALM Flow Chart (IV)



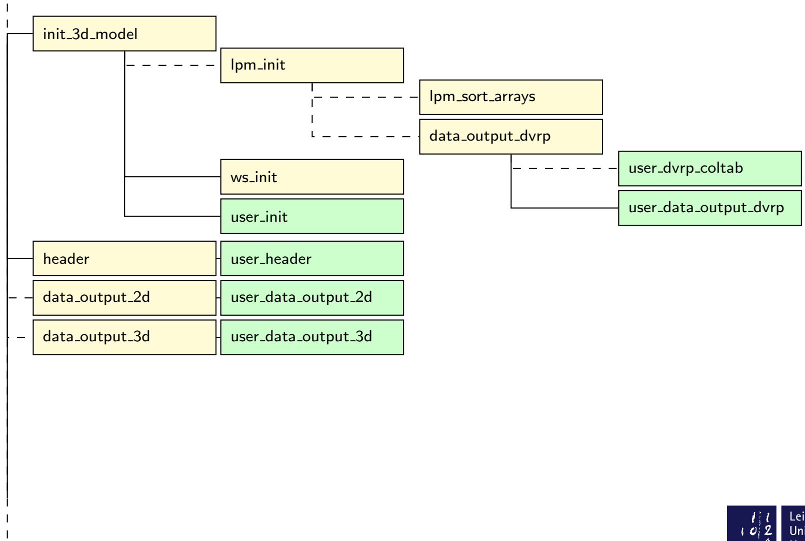
PALM Flow Chart (IV)



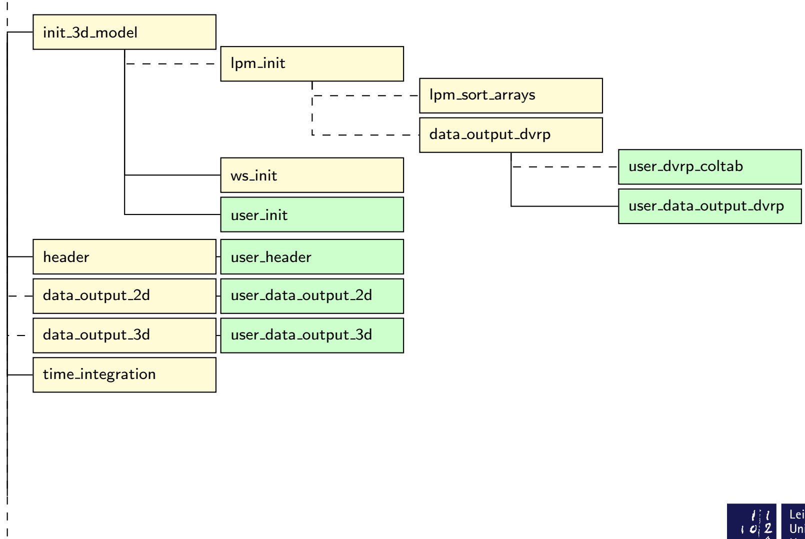
PALM Flow Chart (IV)



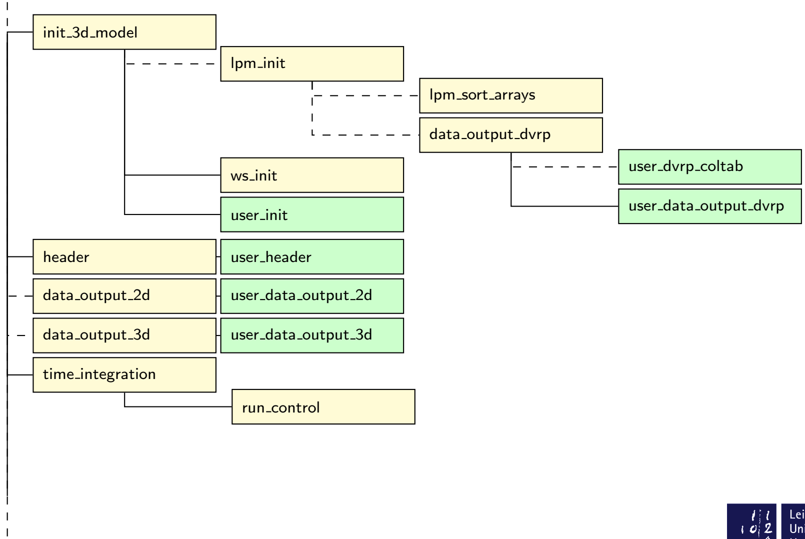
PALM Flow Chart (IV)



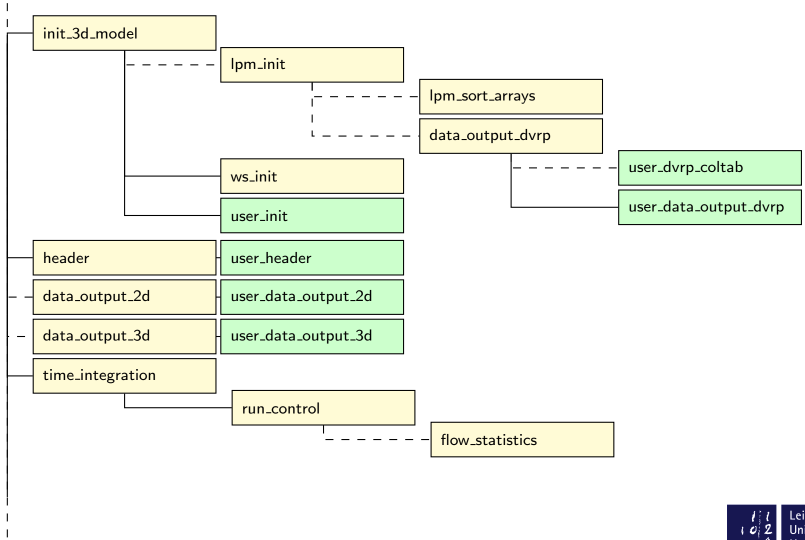
PALM Flow Chart (IV)



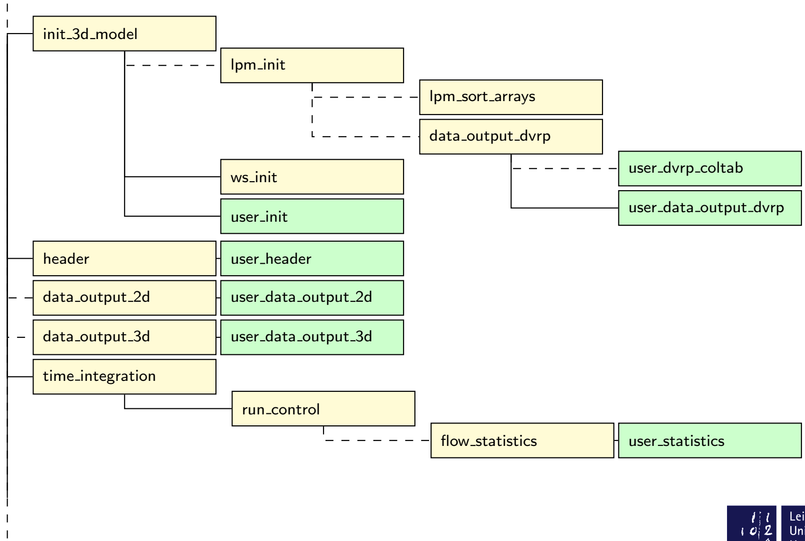
PALM Flow Chart (IV)



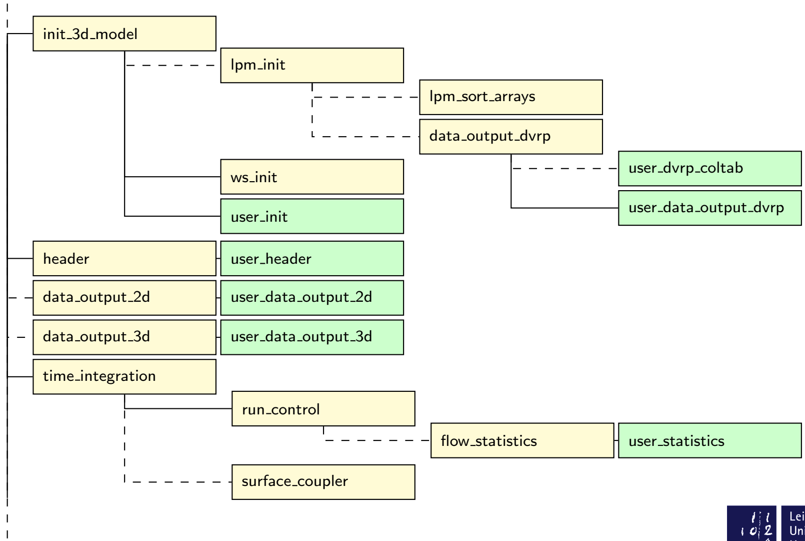
PALM Flow Chart (IV)



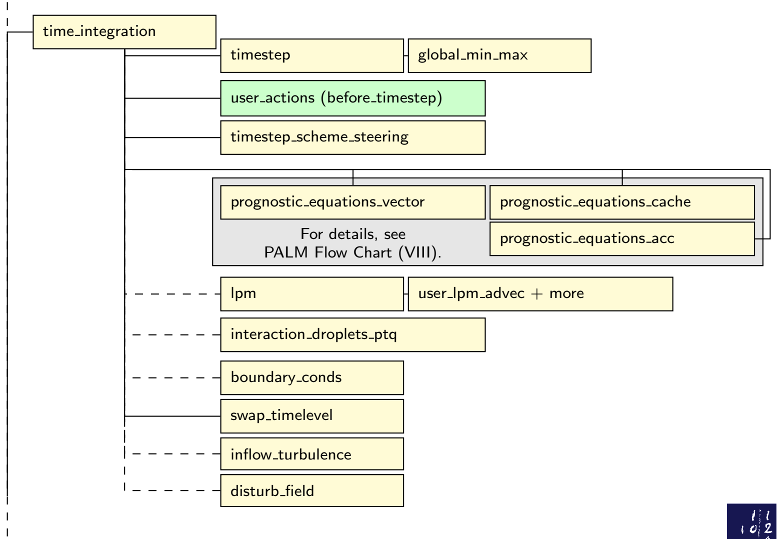
PALM Flow Chart (IV)



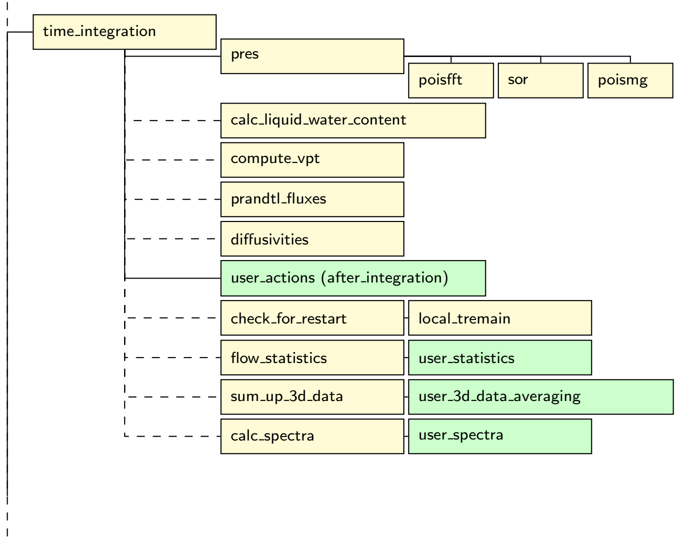
PALM Flow Chart (IV)



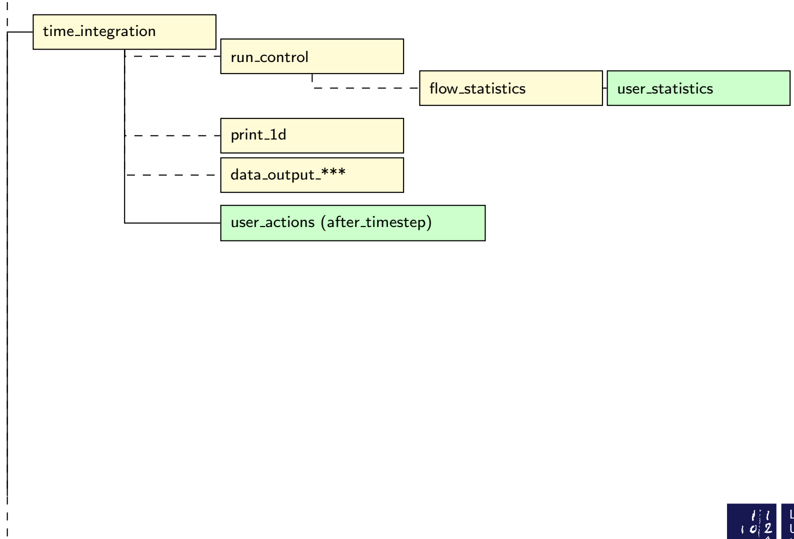
PALM Flow Chart (V)



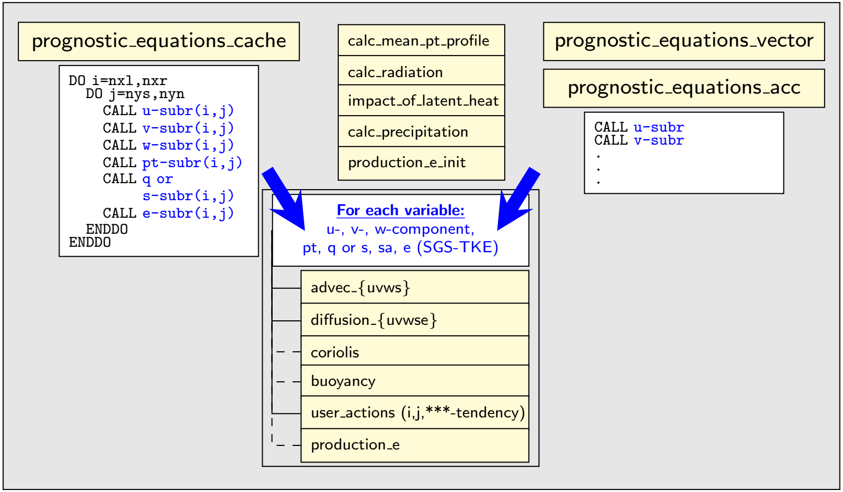
PALM Flow Chart (VI)



PALM Flow Chart (VII)



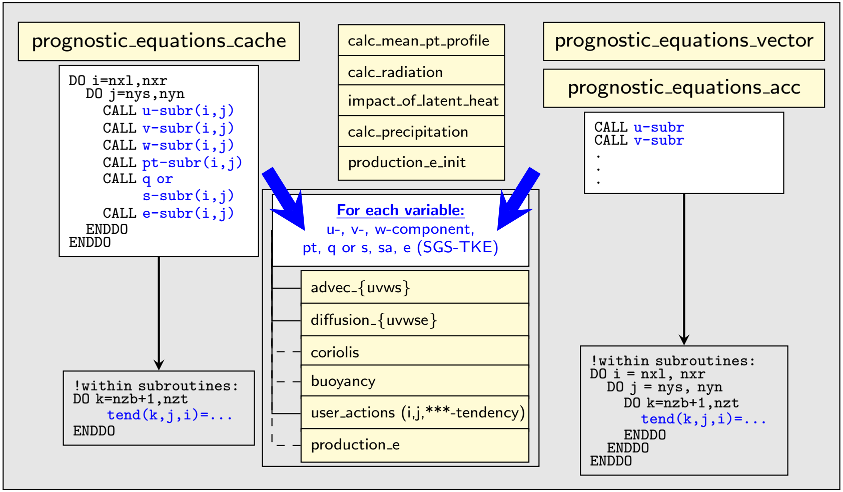
PALM Flow Chart (VIII)



```
DO i=nxl,nxr
  DO j=nys,nyn
    CALL u-subr(i,j)
    CALL v-subr(i,j)
    CALL w-subr(i,j)
    CALL pt-subr(i,j)
    CALL q or
         s-subr(i,j)
    CALL e-subr(i,j)
  ENDDO
ENDDO
```

```
CALL u-subr
CALL v-subr
.
.
.
```

PALM Flow Chart (VIII)



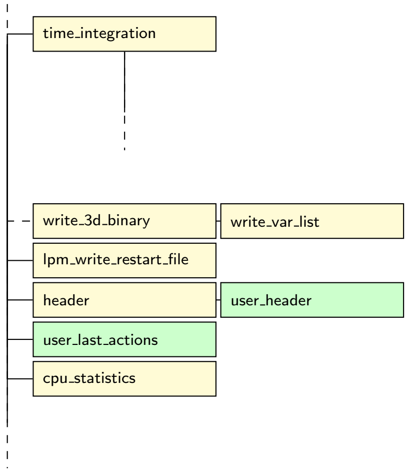
```
DO i=nxl,nxr
  DO j=nys,nyn
    CALL u-subr(i,j)
    CALL v-subr(i,j)
    CALL w-subr(i,j)
    CALL pt-subr(i,j)
    CALL q or
         s-subr(i,j)
    CALL e-subr(i,j)
  ENDDO
ENDDO
```

```
!within subroutines:
DO k=nzb+1,nzt
    tend(k,j,i)=...
ENDDO
```

```
CALL u-subr
CALL v-subr
.
.
.
```

```
!within subroutines:
DO i = nxl, nxr
  DO j = nys, nyn
    DO k=nzb+1,nzt
      tend(k,j,i)=...
    ENDDO
  ENDDO
ENDDO
```

PALM Flow Chart (IX)



Important Variables and Their Declaration

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, e, sa, u_p, v_p, ...

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, e, sa, u_p, v_p, ...
- 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)
nys, nyn (south, north)
nxl, nxr (left, right)
```

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).

Important Variables and Their Declaration

- If only one process/core is used, then

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

Important Variables and Their Declaration

- If only one process/core is used, then

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

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

```
REAL(wp), DIMENSION(:,:,:), POINTER ::
u, u_p, v, v_p, ...
```

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

Important Variables and Their Declaration

- If only one process/core is used, then

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

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

```
REAL(wp), DIMENSION(:,:,:), POINTER ::
u, u_p, v, v_p, ...
```

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

- A pointer free version can be activated with preprocessor-option `-D__nopointer`.

Some Other Frequently Used Variables

variable	index bounds	meaning	comment
zu	nzb:nzt+1	heights of the scalar (u,v) grid levels	zu(0)=-zu(1)
zw	nzb:nzt+1	heights of the w grid level	zw(0)=0
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	
ddx		inverse of dx	ddx=1.0/dx
current_timestep_number		timestep counter	
simulated_time		simulated time in seconds	

Preprocessor Directives (I)

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

```
#if defined( __nopointer )
REAL(wp), DIMENSION(:,:,:), ALLOCATABLE, TARGET :: e, e_p, ...
#else
REAL(wp), DIMENSION(:,:,:), POINTER :: e, e_p, ...
#endif
```

Preprocessor Directives (I)

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

  ```
  #if defined( __nopointer )
  REAL(wp), DIMENSION(:,:,:), ALLOCATABLE, TARGET :: e, e_p, ...
  #else
  REAL(wp), DIMENSION(:,:,:), POINTER :: e, e_p, ...
  #endif
  ```

  If now the compiler is called e.g.

  ```
  ifort -cpp -D __nopointer ... (other options)
  ```

  then the line containing "..., ALLOCATABLE, TARGET :: ..." is compiled.
  If the compiler call is

  ```
  ifort -cpp ... (other options)
  ```

  the line containing "..., POINTER :: ..." is compiled.

Preprocessor Directives (II)

- The preprocessor directives require to include the compiler option "`-cpp`" in any way. Otherwise, the compilation will give error messages. **The option has to be given in the configuration file** `.mrun.config` in the `%cpp_options` line. **Different compilers may require different options!**

Preprocessor Directives (II)

- The preprocessor directives require to include the compiler option "`-cpp`" in any way. Otherwise, the compilation will give error messages. **The option has to be given in the configuration file** `.mrun.config` in the `%cpp_options` line. **Different compilers may require different options!**
- Define-string switches can be combined using logical AND / OR operators && / ||.

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

Preprocessor Directives (III)

In the PALM code, define-string switches are used for following reasons:

Preprocessor Directives (III)

In the PALM code, define-string switches are used for following reasons:

- To switch between system dependent subroutines for:

`__ibm`	IBM-Regatta systems
`__lc`	Linux clusters
`__nec`	NEC-SX systems

Switches are set automatically depending on the host identifier string given with mrun-option "`-h`", eg. "`-h lclocal`" sets "`-D __lc`"

Preprocessor Directives (III)

In the PALM code, define-string switches are used for following reasons:

- To switch between system dependent subroutines for:
  - `__ibm` IBM-Regatta systems
  - `__lc` Linux clusters
  - `__nec` NEC-SX systems

  Switches are set automatically depending on the host identifier string given with mrun-option "`-h`", eg. "`-h lclocal`" sets "`-D __lc`"
- To switch between the serial and the parallel code:
  - `__parallel`

  This switch is set by mrun-option "`-K parallel`".

Preprocessor Directives (IV)

In the PALM code, define-string switches are additionally used for following reasons:

- To enable usage of special software packages which are not included in the compilation process by default
  - `__dvrp_graphics` 3D visualization system (currently out of order)
  - `__spectra` calculation and output of power spectra

  Switches are activated with mrun-option "`-p`",
  eg. "`-p "spectra dvrp_graphics"`"

Preprocessor Directives (IV)

In the PALM code, define-string switches are additionally used for following reasons:

- To enable usage of special software packages which are not included in the compilation process by default
  - `__dvrp_graphics` 3D visualization system (currently out of order)
  - `__spectra` calculation and output of power spectra

  Switches are activated with mrun-option "`-p`",
  eg. "`-p "spectra dvrp_graphics"`"
- To enable special features
  - `__openacc` activates call of external routines required for OpenACC programming
  - `__netcdf`, `__netcdf4`, `__netcdf_parallel` NetCDF I/O with different NetCDF versions

Preprocessor Directives (V)

- Preprocessor directives are also used for string replacement in the code.

  Example:
  A compiler call with preprocessor option

  ```
  ifort -cpp -Dabcd=efgh
  ```

  will replace all strings "abcd" in the code with "efgh" **before** the code is compiled.

  This is used in PALM to change the MPI_REAL datatypes (which are 4 byte long by default), to 8 bytes. The respective cpp-directives are given in the configuration file `.mrun.config.`:

  ```
  %cpp_options -cpp:-DMPI_REAL=MPI_DOUBLE_PRECISION:
               -DMPI_2REAL=MPI_2DOUBLE_PRECISION: ....
  ```