User-defined code

palm group

Institute of Meteorology and Climatology, Leibniz Universität Hannover

User-defined code

Purpose of the user interface

- The standard (default) PALM code cannot account for every specific demand of a user.
- In order to include these specific demands, the user would have to modify the standard code.

Problem:

- New releases of PALM would require the user to add his/her modifications to the new release again.

Solution:

- PALM offers a “user-interface“, i.e. a set of subroutines, where the user can add his/her modifications without changing the standard code, and which can be re-used for future releases of the standard PALM code.
- The user-interface subroutines are almost “empty“ by default. They are called from the standard PALM code but (with some very minor exceptions) do not contain any executable code.
- The user-interface is realized as a module, like other PALM modules.

User-defined code

General structure of the user interface

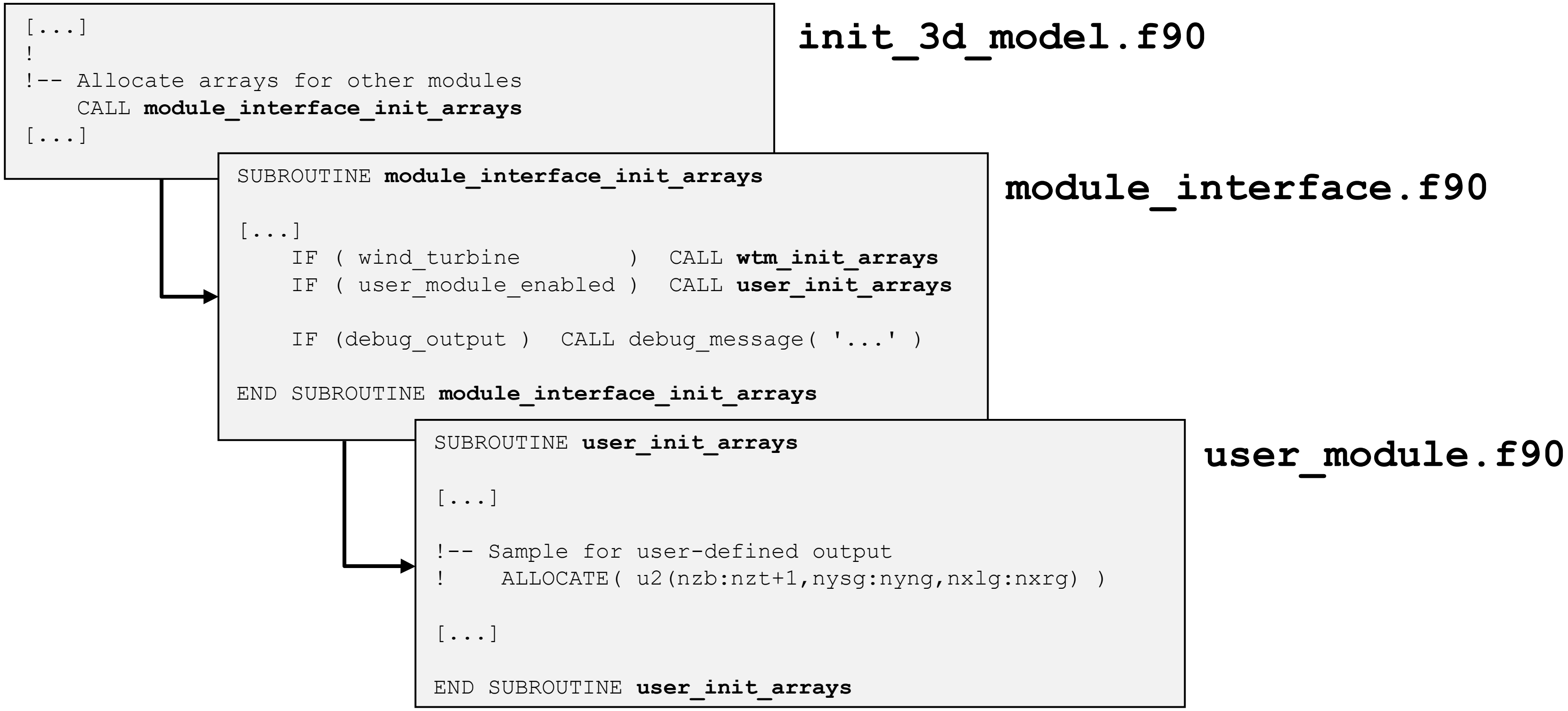
- Most routines can be found within **`.../palm_model_system/packages/palm/model/src/user_module.f90`**.
- Only a few routines have their own files (e.g., **`user_init_radiation.f90`**, **`user_init_flight.f90`**).

```
 MODULE user

    USE arrays_3d
    USE control_parameters
[...]

    IMPLICIT NONE
[...]

    PUBLIC user_parin, user_actions, [...]

    INTERFACE user_parin
       MODULE PROCEDURE user_parin
    END INTERFACE user_parin
    INTERFACE user_actions
       MODULE PROCEDURE user_actions
       MODULE PROCEDURE user_actions_ij
    END INTERFACE user_actions
[...]

    CONTAINS

    SUBROUTINE user_parin
      [...]
    END SUBROUTINE user_parin

[...]
```

User-defined code

Embedding of user-interface routines (I)

- The user-interface routines are called via the module interface at specific locations within the standard PALM code.
- Example for routine **`user_init_arrays`**:



init_3d_model.f90

```
[...]
!
!-- Allocate arrays for other modules
    CALL module_interface_init_arrays
[...]
```

module_interface.f90

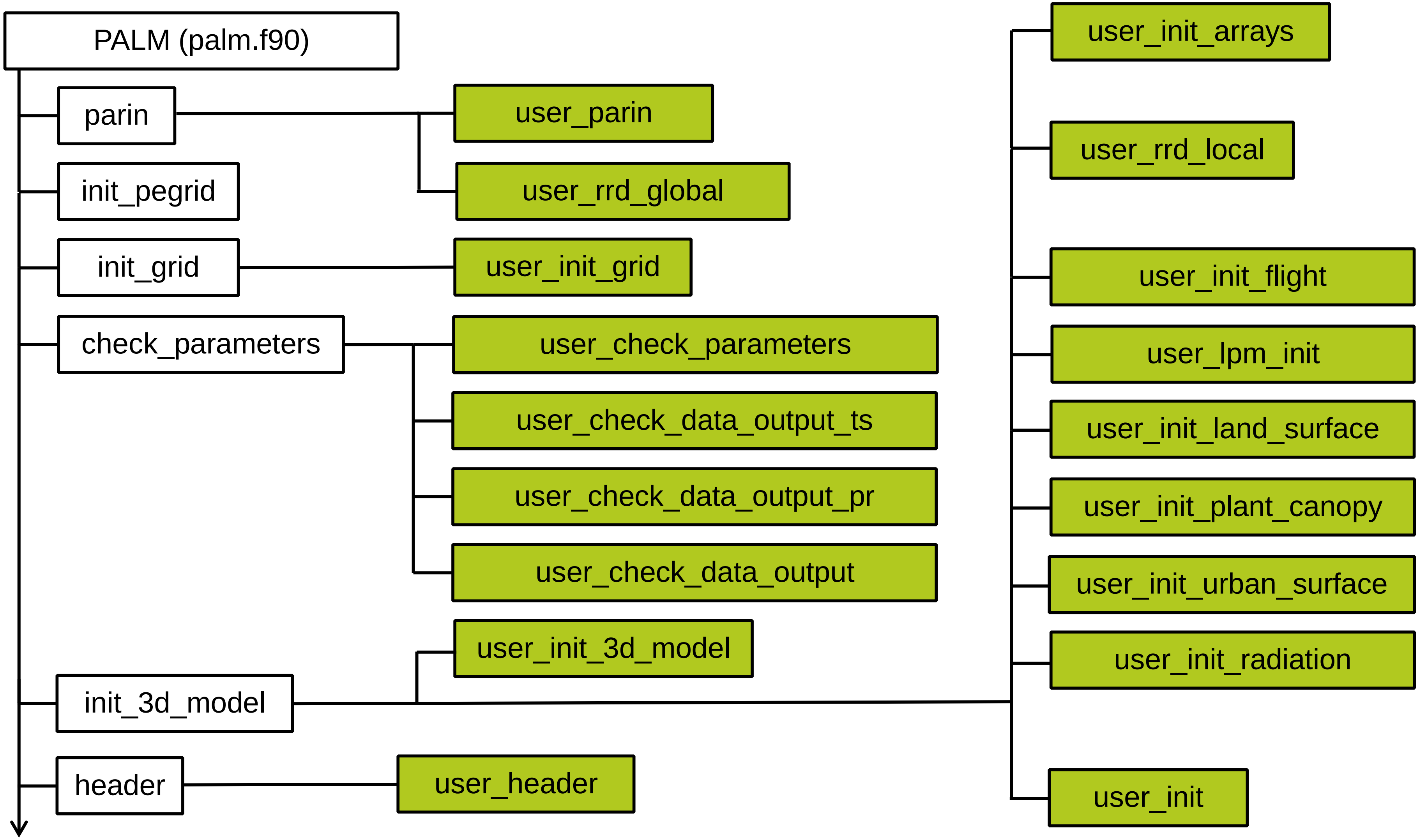
```
SUBROUTINE module_interface_init_arrays

[...]
    IF ( wind_turbine          )  CALL wtm_init_arrays
    IF ( user_module_enabled )  CALL user_init_arrays

    IF (debug_output )  CALL debug_message( '...' )

END SUBROUTINE module_interface_init_arrays
```

user_module.f90

```
SUBROUTINE user_init_arrays

[...]

!-- Sample for user-defined output
!     ALLOCATE( u2(nzb:nzt+1,nysg:nyng,nxlg:nxrg) )

[...]

END SUBROUTINE user_init_arrays
```

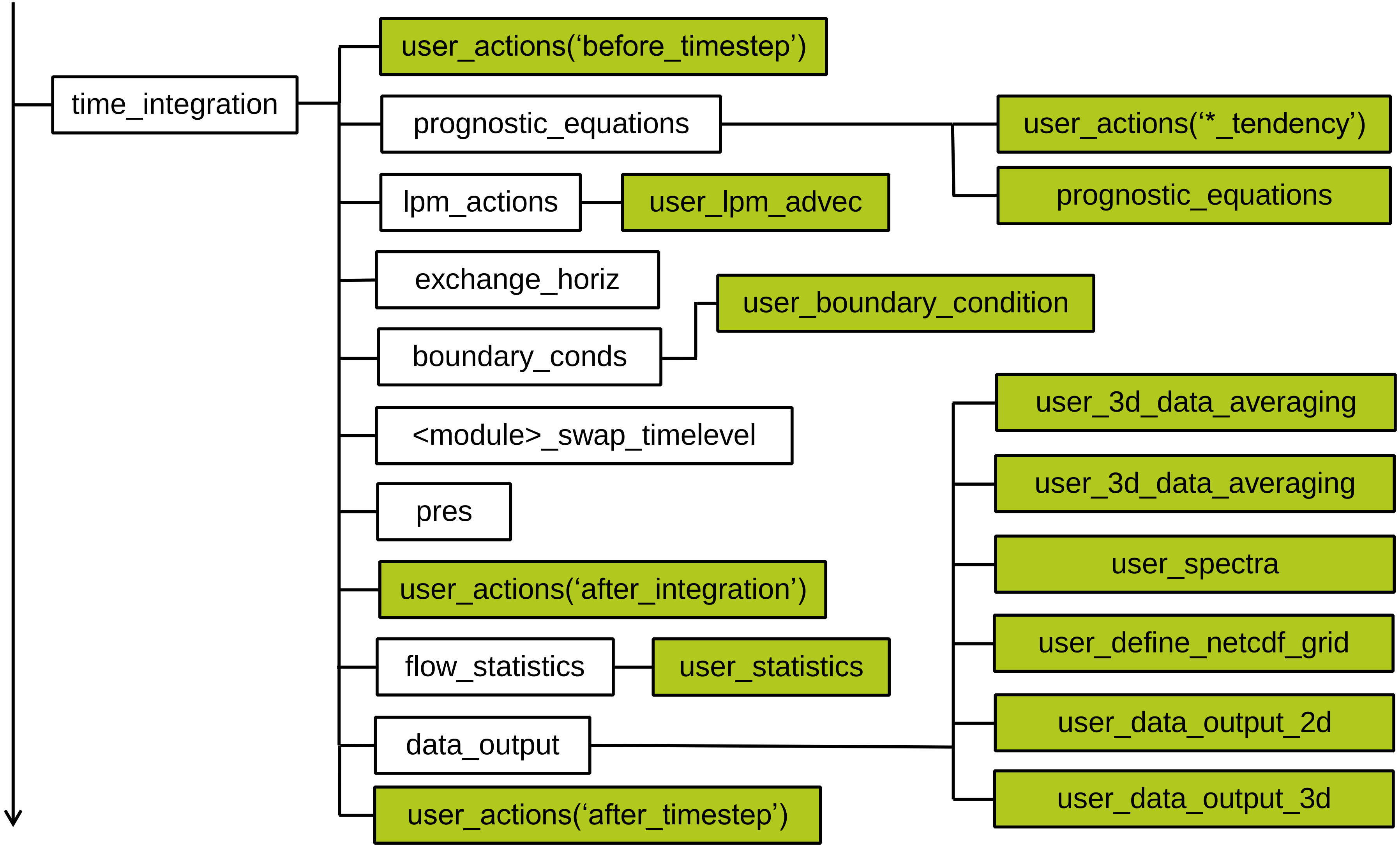
User-defined code

Embedding of user-interface routines (II)
Flow chart overview (I) - Initialization



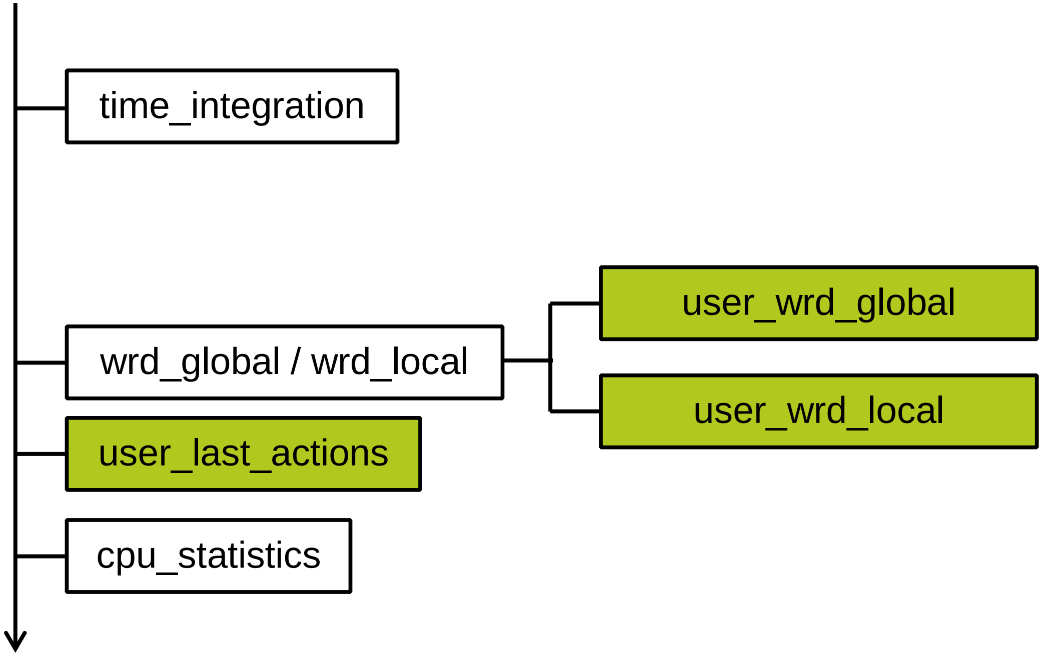
User-defined code

Embedding of user-interface routines (II)
Flow chart overview (II) – Time integration loop



User-defined code

Embedding of user-interface routines (II)
Flow chart overview (III) – Final steps



User-defined code

List of user-interface routines (I)

Name	Arguments	Called from	Task
user_3d_data_averaging	mode, variable	average_3d_data + sum_up_3d_data	temporal averaging for user-defined quantities
user_actions **user_actions_ij**	location i, j, location	time_integration + prognostic_equations	e.g. additional forces to be included in the prognostic equations
user_check_data_output	variable, unit	check_parameters + init_masks	check the user-defined output quantities
user_check_data_output_pr	variable, var_count, unit	check_parameters	check the user-defined profile output quantities
user_check_data_output_ts	variable, var_count, unit	check_parameters	check the user-defined time-series output quantities
user_check_parameters	---	check_parameters	check user-defined variables
user_data_output_2d	av, variable, found, grid, local_pf, two_d	data_output_2d	output/calculation of additional user-defined quantities
user_data_output_3d	av, variable, found, local_pf, nz_do	data_output_3d	output/calculation of additional user-defined quantities
user_data_output_mask	av, variable, found, local_pf	data_output_mask	output of additional masked user-defined quantities
user_define_netcdf_grid	variable, found, grid_x, grid_y, grid_z	netcdf	defining the grid for additional output quantities

User-defined code

List of user-interface routines (II)

Name	Arguments	Called from	Task
user_flight	var, id	virtual_flight_mod	output of flight measurements
user_header	io	header	output user variables to header
user_init	---	init_3d_model	e.g. reading from restart file
user_init_arrays	---	init_3d_model	e.g. reading from restart file
user_init_3d_model	---	init_3d_model	special initializations
user_init_flight	init, k, id, label_leg	virtual_flight_mod	initialization of flight measurements
user_init_grid	gls	init_grid	defining a special topography
user_init_land_surface	---	land_surface_model_mod	initialize land surface model
user_init_plant_canopy	---	init_3d_model	setting of leaf area density and canopy drag coefficient
user_init_radiation	---	radiation_model_mod	initialize radiation model
user_init_urban_surface	---	urban_surface_mod	initialize urban surface model
user_last_actions	---	palm	e.g. output for restart runs
user_lpm_advec	ip, jp, kp	lpm	modification of initial particles
user_lpm_init	---	lpm	modification of initial particles

User-defined code

List of user-interface routines (III)

Name	Arguments	Called from	Task
MODULE user (user_module.f90)	---	---	contains user defined variables and routines
user_parin		parin	reading user variables
user_prognostic_equations	i, j, i_omp_start, tn	prognostic_equations	prognostic equation for user-defined quantity
user_rrd_global **user_rrd_local**	i, nxlfa, nxl_on_file, nxrfa, nxr_on_file, nynfa, nyn_on_file, nysfa, nys_on_file, offset_xa, offset_ya, overlap_count, tmp_2d, tmp_3d	rrd_global rrd_local	reading user-defined 2d/3d-arrays from the restart file
user_spectra	mode, m, pr	data_output_spectra	output/calculation of additional user-defined quantities
user_statistics	mode, sr, tn	flow_statistics	calculating additional horizontal averages + time series quantities
user_wrd_global **user_wrd_local**	---	wrd_global wrd_local	writing user-defined 2d/3d-arrays into the restart file

See PALM online documentation under
http://palm-model.org/trac/wiki/doc/app/userint/int for detailed explanations.

User-defined code

Data access / exchange

- **User-interface access to default PALM code data:**
  - By including the respective PALM modules in the user-interface subroutines.

```
SUBROUTINE user_init_flight( init, k, id, label_lag )

   USE control_parameters

   USE indices

   USE kinds

   USE user

   IMPLICIT NONE

   CHARACTER(LEN=10), OPTIONAL ::  label_leg     !< label of the leg

   INTEGER(iwp), OPTIONAL                ::  id  !< variable id
   INTEGER(iwp), OPTIONAL, INTENT(INOUT) ::  k   !< index of variable

   LOGICAL ::  init  !< true for initial call

[...]
```

- **Within the user-interface:**
  - By the module **user** (**user_module.f90**).

User-defined code

Usage of user_actions (I)

- **`user_actions`** is designed to add additional terms to the prognostic equations or to carry out special actions at the beginning or the end of each timestep.
- Several calls of **`user_actions`** (via **`module_interface_actions`**) can be found within **`time_integration`** and **`prognostic_equations`**. The place, from which it is called, is communicated to the routine by a string-argument, e.g.

  `CALL module_interface_actions( 'u-tendency' ).`

- This call means that it originates from a line within **`prognostic_equations`**, where the tendencies for the u-component are calculated and integrated:

```
[...]

    CALL user_actions( 'u-tendency' )

!
!-- Prognostic equation for u-velocity component
    DO  i = nxlu, nxr
       DO  j = nys, nyn
          DO  k = nzb+1, nzt
             u_p(k,j,i) = u(k,j,i)                                                   &
                        + ( dt_3d * ( tsc(2) * tend(k,j,i) + tsc(3) * tu_m(k,j,i) )  &
                        - tsc(5) * rdf(k) * ( u(k,j,i) - u_init(k) ) &
[...]
```

User-defined code

Usage of user_actions (II)

- Additional tendencies can be included by the user at the respective code line in user_actions:

```
 SUBROUTINE user_actions( location )

[...]

!--    Here the user-defined actions follow. No calls for single grid points are allowed at  &
!--    locations before and after the timestep, since these calls are not within an i,j-loop
       SELECT CASE ( location )

         CASE ( 'before_timestep' )
!
!--            Enter actions to be done before every timestep here

[...]

         CASE ( 'u-tendency' )
!
!--            Enter actions to be done in the u-tendency term here
               DO  i = nxl, nxr
                  DO  j = nys, nyn
                     DO  k = nzb+1, nzt
                        tend(k,j,i) = tend(k,j,i) - const * u(k,j,i) ...
                     ENDDO
                  ENDDO
               ENDDO

          CASE ( 'v-tendency’ )

[...]
```

User-defined code

Usage of user_actions (III)

- The different versions of **prognostic_equations** (**_cache**, and **_vector**) contain different calls of **user_actions**:
  - From **prognostic_equations_vector**:

    CALL user_actions('u-tendency').
  - From **prognostic_equations_cache**:

    CALL user_actions(i,j,'u-tendency').
- In case that **prognostic_equations_cache** is used, the user has to add code in the interface routine **user_actions_ij**.
- Here, only the **k**-loop (vertical direction) has to be used, because loops over **i** and **j** are carried out in **prognostic_equations_cache**.

```
 SUBROUTINE user_actions_ij( i, j, location )

[...]

!
!--    Here the user-defined actions follow
       SELECT CASE ( location )

[...]

         CASE ( 'u-tendency' )
             DO  k = nzb+1, nzt-1
                 tend(k,j,i) = tend(k,j,i) + ...
             ENDDO

         CASE ( 'v-tendency' )

[...]
```

User-defined code

Steering the user_interface

- For steering the user-interface code, the user can add some additional variables and set their respective values within the parameter-file (**<run identifier>_p3d**). This requires the following actions (example for a variable named **foo**):

(1) Add the variable name to module **user** in order to define it and to make it available in all user-interface subroutines. Set a default value for this variable.

```
MODULE user
[...]
REAL(wp)     ::  foo = 0.0_wp
```

(2) Add the variable to the NAMELIST **/user_parameters/**. This NAMELIST already contains five predefined variables.

```
SUBROUTINE user_parin
[...]

   NAMELIST /user_parameters/  data_output_masks_user, data_output_pr_user, &
                  data_output_user, region, switch_off_module, foo

[...]
END SUBROUTINE user_parin
```

(3) Add the NAMELIST **&user_parameters** to the parameter file (**<run identifier>_p3d**) and assign a value to this variable.

```
&user_parameters
  foo = 12345.6,
/
```

(4) Output the variable's value using the routine **user_header**.

User-defined code

User-defined output

- A typical request of users is the calculation and output of quantities which are not part of PALM's standard output (e.g. a 3D-array of the resolved-scale vertical heat flux).
- The default user interface includes a number of subroutines which allow the calculation of user-defined quantities and output of these quantities as profiles, timeseries, 2d cross section, or 3d volume data. These are e.g.

  `user_check_data_output`, `user_check_data_output_pr`, `user_define_netcdf_grid`, `user_statistics`, `user_3d_data_averaging`, `user_data_output_2d`, `user_data_output_3d`.

- The respective subroutines contain examplary code lines (written as comment lines) for calculating and output exemplary quantities.
- These quantities are output to PALM‘s standard NetCDF files, i.g.

  `<run identifier>_pr.000.nc`, `<run identifier>_ts.000.nc`, `<run identifier>_xy.000.nc`, or **`<run_identifier>_3d.000.nc`.**

- The online documentation gives very detailed instructions about how to modify the interface in order to output user-defined quantities under

  http://palm-model.org/trac/wiki/doc/app/userint/output.

User-defined code

User-defined data for restart runs (I)

- It might be required to save the values of user-defined variables at the end of a model run in order to use them for a restart run.
- This can be realized using the routine `user_wrd_local`.
- `'14'` is the file-id for the restart file in Fortran binary format (local filename **`BINOUT`**).

```
 SUBROUTINE user_wrd_local
 [...]
    IF ( TRIM( restart_data_format_output ) == 'fortran_binary' )  THEN

       IF ( ALLOCATED( user_array1 ) )  THEN
          CALL wrd_write_string( 'user_array1' )
          WRITE ( 14 )  user_array1
       ENDIF
       IF ( ALLOCATED( user_array2 ) )  THEN
          CALL wrd_write_string( 'user_array2' )
          WRITE ( 14 )  user_array2
       ENDIF

    ELSEIF ( restart_data_format_output(1:3) == 'mpi' )  THEN

       IF ( ALLOCATED( user_array_1 ) ) CALL wrd_mpi_io( 'user_array1', user_array1 )
       IF ( ALLOCATED( user_array_2 ) ) CALL wrd_mpi_io( 'user_array2', user_array2 )

    ENDIF
[...]
```

User-defined code

User-defined data for restart runs (II)

- Additionally, these variables have to be read from the restart file (file-id '13', local filename BININ) by adding code to the routine **user_rrd_local_ftn**:

```
 SUBROUTINE user_rrd_local_ftn( i, k, nxlf, nxlc, nxl_on_file, nxrf, nxrc,      &
                                nxr_on_file, nynf, nync, nyn_on_file, nysf,     &
                                nysc, nys_on_file, tmp_3d, found )
[...]

    found = .TRUE.

    SELECT CASE ( restart_string(1:length) )

       CASE ( 'user_array1' )
          IF ( .NOT. ALLOCATED( user_array1 ) ) THEN
             ALLOCATE( user_array1(nzb:nzt+1,nysg:nyng,nxlg:nxrg) )
          ENDIF
          IF ( k == 1 )  READ ( 13 )  tmp_3d
          user_array1(:,nysc-nbgp:nync+nbgp,nxlc-nbgp:nxrc+nbgp) &
              = tmp_3d(:,nysf-nbgp:nynf+nbgp,nxlf-nbgp:nxrf+nbgp)

[...]

       CASE DEFAULT
          found = .FALSE.

    END SELECT

 END SUBROUTINE user_rrd_local_ftn
```

User-defined code

User-defined data for restart runs (III)

- If the restart file is created using MPI format, these variables have to be read from the restart file by adding code to the routine **user_rrd_local_mpi**:

```
 SUBROUTINE user_rrd_local_mpi
[...]

    CALL rd_mpi_io_check_array( 'user_array1' , found = array_found )
    IF ( array_found )  THEN
       IF ( .NOT. ALLOCATED( user_array1 ) ) &
           ALLOCATE( user_array1(nzb:nzt+1,nysg:nyng,nxlg:nxrg) )
       CALL rrd_mpi_io( 'user_array1', user_array1 )
    ENDIF

    CALL rd_mpi_io_check_array( 'user_array2' , found = array_found )
    IF ( array_found )  THEN
       IF ( .NOT. ALLOCATED( user_array2 ) ) &
           ALLOCATE( user_array2(nzb:nzt+1,nysg:nyng,nxlg:nxrg) )
       CALL rrd_mpi_io( 'user_array2', user_array2 )
    ENDIF

 END SUBROUTINE user_rrd_local_mpi
```

User-defined code

Using the user-interface with palmrun (I)

- Users can add their own (modified) user-interface to a PALM run by carrying out the following steps:

1. Copy the needed default (empty) user-interface files (e.g. user_module.f90, user_init_grid.f90) to a USER_CODE directory within the desired run-identifier structure, e.g.:

```
cd ~/palm/current_version
mkdir JOBS/example_cbl/USER_CODE
cp {source_path}/user_module.f90    JOBS/example_cbl/USER_CODE
cp {source_path}/user_init_grid.f90 JOBS/example_cbl/USER_CODE
```

2. Modify the interface routines accordingly.

3. Start a a PALM run by executing

```
palmrun -r example_cbl...
```

The files user_*.f90 will be automatically compiled within the job/interactive run and will replace the respective PALM default user-interface files.

User-defined code

Using the user-interface with palmrun (II)

- The modified user-interface file cannot be pre-compiled by using **`palmbuild`**!
- Compilation of the user-interface can be very time consuming. Use **`palmrun`**-option **`-V`** to re-use user-interface routines that have been compiled previously for the specific run-identifier.
- PALM‘s user-interface mechanism allows to use different interfaces for different runs via their run-identifier. Therefore, users may store the respective interface-files in subdirectories, e.g.

  `JOBS/run_x/USER_CODE`, and **`JOBS/run_y/USER_CODE`**.

- If palmrun gets started with a specific run-identifier, the corresponding interface will be used.