The PALM User-Interface

PALM group

Institute of Meteorology and Climatology, Leibniz Universität Hannover

last update: 21st September 2015

Purpose of the User-Interface

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.

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 (current release is 3.10) would require the user to add his modifications to the new release again.

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 (current release is 3.10) would require the user to add his modifications to the new release again.

Solution:

- PALM offers a “user-interface“, i.e. a set of subroutines, where the user can add his modifications, and which can be re-used for future releases of the standard PALM 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 (current release is 3.10) would require the user to add his modifications to the new release again.

Solution:

- PALM offers a “user-interface“, i.e. a set of subroutines, where the user can add his modifications, and which can be re-used for future releases of the standard PALM code.
- By using the user-interface, the standard code does not have to be modified by the user in most of the cases.

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 (current release is 3.10) would require the user to add his modifications to the new release again.

Solution:

- PALM offers a “user-interface“, i.e. a set of subroutines, where the user can add his modifications, and which can be re-used for future releases of the standard PALM code.
- By using the user-interface, the standard code does not have to be modified by the user in most of the cases.
- 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.

General Structure of the User-Interface

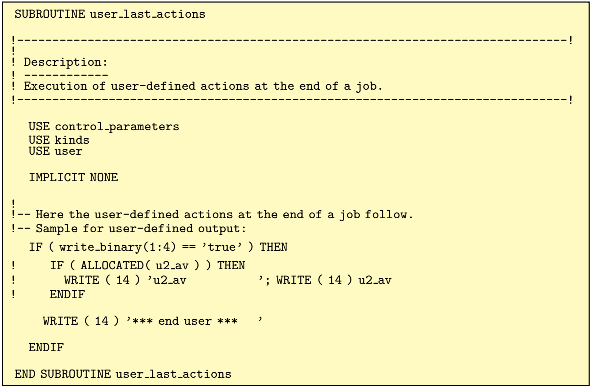
- All routines can be found under `.../trunk/SOURCE`.

General Structure of the User-Interface

- All routines can be found under `.../trunk/SOURCE`.
- There is one file for each routine. Filenames are `user_*.f90`.
  Example: `user_last_actions.f90`

General Structure of the User-Interface

- All routines can be found under `.../trunk/SOURCE`.
- There is one file for each routine. Filenames are `user_*.f90`.
  Example: `user_last_actions.f90`



```
 SUBROUTINE user_last_actions

!------------------------------------------------------------------------------!
!
! Description:
! ------------
! Execution of user-defined actions at the end of a job.
!------------------------------------------------------------------------------!

    USE control_parameters
    USE kinds
    USE user

    IMPLICIT NONE

!
!-- Here the user-defined actions at the end of a job follow.
!-- Sample for user-defined output:
    IF ( write_binary(1:4) == 'true' ) THEN
!      IF ( ALLOCATED( u2_av ) ) THEN
!         WRITE ( 14 ) 'u2_av           '; WRITE ( 14 ) u2_av
!      ENDIF

       WRITE ( 14 ) '*** end user ***    '

    ENDIF

 END SUBROUTINE user_last_actions
```

Embedding of User-Interface Routines

- The user-interface routines are called from specific, well-defined locations in the standard PALM code.

  Example from `palm.f90`:

Embedding of User-Interface Routines

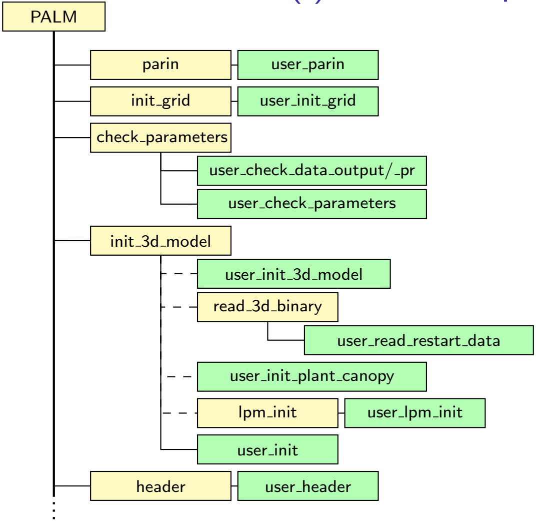
- The user-interface routines are called from specific, well-defined locations in the standard PALM code.

Example from `palm.f90`:

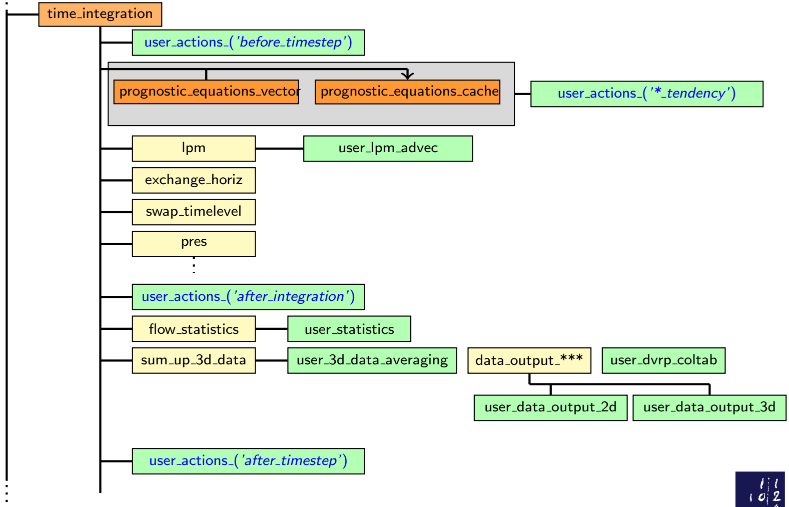
```
 ...
!
!-- If required, final user-defined actions, and
!-- last actions on the open files and close files. Unit 14 was opened
!-- in write_3d_binary but it is closed here, to allow writing on this
!-- unit in routine user_last_actions.
   CALL cpu_log( log_point(4), 'last actions', 'start' )
   DO i = 0, io_blocks-1
     IF ( i == io_group ) THEN
       CALL user_last_actions
       IF ( write_binary(1:4) == 'true' ) CALL close_file( 14 )
     ENDIF
#if defined( __parallel )
     CALL MPI_BARRIER( comm2d, ierr )
#endif
   ENDDO
   CALL close_file( 0 )
   CALL close_dvrp
   CALL cpu_log( log_point(4), 'last actions', 'stop' )
 ...
!
!-- Take final CPU-time for CPU-time analysis
   CALL cpu_log( log_point(1), 'total', 'stop' )
   CALL cpu_statistics
#if defined( __parallel )
   CALL MPI_FINALIZE( ierr )
#endif

 END PROGRAM palm
```

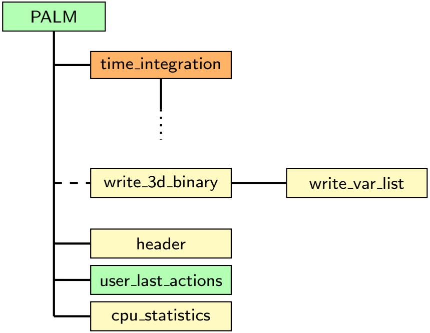
Embedding of User-Interface Routines
Flow Chart Overview (I): Initial Steps



Embedding of User-Interface Routines
Flow Chart Overview (II): Time Integration Loop



Embedding of User-Interface Routines
Flow Chart Overview (III): Final Steps



PALM

time_integration

write_3d_binary

write_var_list

header

user_last_actions

cpu_statistics

Complete 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	location i, j, location	time_integration + prognostic_equations	e.g. additional forces to be included in the prognostic equations
user_dummy (user_additional_routines.f90)	- - -	- - -	for additional subroutines defined by the user
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_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_dvrp	output_variable, local_pf	data_output_dvrp	output 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_dvrp_coltab	mode, variable	data_output_dvrp	defining color tables for particles
user_header	io	header	output user variables to header
user_init	- - -	init_3d_model	e.g. reading from restart file

Complete List of User-Interface Routines (II)

Name	Arguments	Called from	Task
user_init_3d_model	- - -	init_3d_model	special initializations
user_init_grid	gls	init_grid	defining a special topography
user_init_plant_canopy	- - -	init_3d_model	setting of leaf area density and canopy drag coefficient
user_last_actions	- - -	palm	e.g. output for restart runs
user_lpm_advec	- - -	lpm	modification of particles after advection
user_lpm_init	- - -	lpm_init	defining initial particle sources
user_lpm_set_attributes	- - -	lpm	defining particles attributes
MODULE user (user_module.f90)	- - -	- - -	contains user defined variables
user_parin		parin	reading user variables
user_read_restart_data	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	read_3d_binary	reading user-defined 2d/3d-arrays from the restart file
user_spectra	mode, m, pr	calc_spectra + data_output_spectra	output/calculation of additional user-defined quantities
user_statistics	mode, sr, tn	flow_statistics	calculating additional horizontal averages + time series quantities

See PALM online documentation under
http://palm.muk.uni-hannover.de/trac/wiki/doc/app/userint
for detailed explanations.

Data Access / Exchange

Data Access / Exchange

- **Between the standard PALM code and the user-interface:**

Data Access / Exchange

- **Between the standard PALM code and the user-interface:**
  - by including the respective PALM modules in the user-interface subroutines.

```
SUBROUTINE user_actions( location )

   USE arrays_3d
   USE control_parameters
   USE cpulog
   USE indices
   USE interfaces
   USE pegrid
   USE user

   IMPLICIT NONE

   CHARACTER (LEN=*) :: location

   INTEGER :: i, j, k
```

Data Access / Exchange

- **Between the standard PALM code and the user-interface:**
  - by including the respective PALM modules in the user-interface subroutines.

```
SUBROUTINE user_actions( location )

   USE arrays_3d
   USE control_parameters
   USE cpulog
   USE indices
   USE interfaces
   USE pegrid
   USE user

   IMPLICIT NONE

   CHARACTER (LEN=*) :: location

   INTEGER :: i, j, k
```

- **Within the user-interface:**
  - by the module `user` (file `user_module.f90`), which is used in every subroutine included in the interface.
    This module is (and should be) never used in the standard PALM code (otherwise, the default code would depend on the user interface).

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.

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.
- Therefore, several calls of user_actions can be found in the default PALM routines time_integration and prognostic_equations. The place from which it is called is communicated to the routine by a string-argument, e.g.

```
CALL user_actions( 'u-tendency' )
```

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.
- Therefore, several calls of user_actions can be found in the default PALM routines time_integration and prognostic_equations. The place from which it is called is communicated to the routine by a string-argument, e.g.

```
CALL user_actions( 'u-tendency' )
```

It means that this call is from a line within prognostic_equations, where the tendencies for the u-component are calculated and integrated:

```
DO i = nxl, nxr
  DO j = nys, nyn
...
    CALL diffusion_u( i, j )
    CALL coriolis( i, j, 1 )
...
    CALL user_actions( i, j, 'u-tendency' )
!
!-- Prognostic equation for u-velocity component
    DO k = nzb_u_inner(j,i)+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) - ug(k) )
    ENDDO
...
```

Usage of user_actions (II)

- Additional tendencies have to 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 ( 'after_timestep' )
!
!--         Enter actions to be done after 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 = nxb+1, nzt
                 tend(k,j,i) = tend(k,j,i) - const * u(k,j,i) ...
             ENDDO
           ENDDO
         ENDDO


      CASE ( 'v-tendency' )
...
```

Usage of user_actions (III)

- The different versions of prognostic_equations (prognostic_equations_cache, prognostic_equations_vector) contain different calls of user_actions:

Usage of user_actions (III)

- The different versions of prognostic_equations (prognostic_equations_cache, prognostic_equations_vector) contain different calls of user_actions:
- From prognostic_equations_vector: `CALL user_actions( 'u-tendency' )`

Usage of user_actions (III)

- The different versions of prognostic_equations (prognostic_equations_cache, prognostic_equations_vector) contain different calls of user_actions:
- From prognostic_equations_vector: `CALL user_actions( 'u-tendency' )`
- From prognostic_equations, prognostic_equations_cache: `CALL user_actions( i, j, 'u-tendency' )`

Usage of user_actions (III)

- The different versions of prognostic_equations (prognostic_equations_cache, prognostic_equations_vector) contain different calls of user_actions:
- From prognostic_equations_vector: CALL user_actions('u-tendency')
- From prognostic_equations, prognostic_equations_cache: CALL user_actions(i, j, 'u-tendency')
- In case that prognostic_equations prognostic_equations_cache are used, the user has to add his code in the interface routine user_actions_ij:

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

    USE control_parameters
    USE pegrid
    USE user

    IMPLICIT NONE

    CHARACTER (LEN=*) :: location

    INTEGER(iwp) :: i, idum, j

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

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

      CASE ( 'v-tendency' )
```

Usage of user_actions (III)

- The different versions of prognostic_equations (prognostic_equations_cache, prognostic_equations_vector) contain different calls of user_actions:
- From prognostic_equations_vector: CALL user_actions('u-tendency')
- From prognostic_equations, prognostic_equations_cache: CALL user_actions(i, j, 'u-tendency')
- In case that prognostic_equations prognostic_equations_cache are used, the user has to add his 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 executed in prognostic_equations_cache.

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

    USE control_parameters
    USE pegrid
    USE user

    IMPLICIT NONE

    CHARACTER (LEN=*) :: location

    INTEGER(iwp) :: i, idum, j

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

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

      CASE ( 'v-tendency' )
```

Steering the User-Interface

For steering the user-interface code, the user may want to add some additional variables and set their respective values within the parameter-file (e.g. example_cbl_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
...
 END MODULE user
```

Steering the User-Interface

For steering the user-interface code, the user may want to add some additional variables and set their respective values within the parameter-file (e.g. `example_cbl_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
...
 END MODULE user
```

2. Add the variable to the NAMELIST `/userpar/`. This NAMELIST already contains four predefined variables.

```
 SUBROUTINE user_parin
...
    NAMELIST  /userpar/   data_output_pr_user, data_output_user,
                                      foo, region
...
 END SUBROUTINE user_parin
```

Steering the User-Interface

For steering the user-interface code, the user may want to add some additional variables and set their respective values within the parameter-file (e.g. `example_cbl_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
...
 END MODULE user
```

2. Add the variable to the NAMELIST `/userpar/`. This NAMELIST already contains four predefined variables.

```
 SUBROUTINE user_parin
...
    NAMELIST  /userpar/   data_output_pr_user, data_output_user,
                                 foo, region
...
 END SUBROUTINE user_parin
```

3. Add the NAMELIST `&userpar` to the parameter file (e.g. `example_cbl_p3d`) and assign a value to this variable.

```
 &inipar    nx = ...   /
 &d3par    end_time = 3600.0, ...   /
 &userpar   foo = 12345.6   /
```

Steering the User-Interface

For steering the user-interface code, the user may want to add some additional variables and set their respective values within the parameter-file (e.g. example_cbl_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
...
END MODULE user
```

2. Add the variable to the NAMELIST /userpar/. This NAMELIST already contains four predefined variables.

```
SUBROUTINE user_parin
...
   NAMELIST  /userpar/   data_output_pr_user, data_output_user,
                                foo, region
...
END SUBROUTINE user_parin
```

3. Add the NAMELIST &userpar to the parameter file (e.g. example_cbl_p3d) and assign a value to this variable.

```
&inipar   nx = ...   /
&d3par   end_time = 3600.0, ...   /
&userpar  foo = 12345.6  /
```

4. Output the variable's value using user_header.

User-Defined Output

User-Defined Output

- A very 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 (temperature) flux).

User-Defined Output

- A very 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 (temperature) 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`.

User-Defined Output

- A very 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 (temperature) 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, as an example, code lines (written as comment lines) for calculating and output the square of the u-component velocity.

User-Defined Output

- A very 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 (temperature) 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, as an example, code lines (written as comment lines) for calculating and output the square of the u-component velocity.
- These quantities are output to PALM‘s standard netCDF files, i.e. `DATA_1D_PR_NETCDF`, `DATA_1D_TS_NETCDF`, `DATA_2D_XY_NETCDF` or `DATA_3D_NETCDF`.

User-Defined Output

- A very 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 (temperature) 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, as an example, code lines (written as comment lines) for calculating and output the square of the u-component velocity.
- These quantities are output to PALM‘s standard netCDF files, i.e. `DATA_1D_PR_NETCDF`, `DATA_1D_TS_NETCDF`, `DATA_2D_XY_NETCDF` or `DATA_3D_NETCDF`.
- The online documentation gives very detailed instructions about how to modify the interface in order to output user-defined quantities under

http://palm.muk.uni-hannover.de/trac/wiki/doc/app/userint/output

User-Defined Data for Restart Runs (I)

- It might be neccessary 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 done using the routine `user_last_actions`.
  "`14`" is the file-id for the restart file (local filename `BINOUT`):

```
SUBROUTINE user_last_actions
...
   WRITE ( 14 ) 'foo                    '; WRITE ( 14 ) foo
   WRITE ( 14 ) 'bar                    '; WRITE ( 14 ) bar

   WRITE ( 14 ) '*** end user ***       '

END SUBROUTINE user_last_actions
```

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_read_restart_data:

```
SUBROUTINE user_read_restart_data( 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 )
...
   IF ( initializing_actions == 'read_restart_data' ) THEN
      READ ( 13 ) field_char
      DO WHILE ( TRIM( field_char ) '*** end user ***' )

         nxlf = nxlfa(i,k) ...
         SELECT CASE ( TRIM( field_char ) )

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

      ENDDO

      READ ( 13 ) field_char

      ENDDO
   ENDIF

END SUBROUTINE user_read_restart_data
```

Using the User-Interface with mrun

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

Using the User-Interface with mrun

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

1. Copy the default (empty) user-interface files that you need (e.g. user_module.f90, user_parin.f90, user_actions.f90) to a directory of your choice, e.g.:

```
cd ~/palm/current_version
mkdir -p USER_CODE/example_cbl
cp trunk/SOURCE/user_module.f90 USER_CODE/example_cbl/user_module.f90
cp trunk/SOURCE/user_parin.f90  USER_CODE/example_cbl/user_parin.f90
...
```

Using the User-Interface with mrun

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

1. Copy the default (empty) user-interface files that you need (e.g. user_module.f90, user_parin.f90, user_actions.f90) to a directory of your choice, e.g.:

```
cd ~/palm/current_version
mkdir -p USER_CODE/example_cbl
cp trunk/SOURCE/user_module.f90 USER_CODE/example_cbl/user_module.f90
cp trunk/SOURCE/user_parin.f90  USER_CODE/example_cbl/user_parin.f90
...
```

2. Set an additional path in the configuration file `.mrun.config` to allow `mrun` to find and include this file:

```
%add_source_path    $base_directory/USER_CODE/$fname
```

Using the User-Interface with mrun

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

1. Copy the default (empty) user-interface files that you need (e.g. user_module.f90, user_parin.f90, user_actions.f90) to a directory of your choice, e.g.:

```
cd ~/palm/current_version
mkdir -p USER_CODE/example_cbl
cp trunk/SOURCE/user_module.f90 USER_CODE/example_cbl/user_module.f90
cp trunk/SOURCE/user_parin.f90  USER_CODE/example_cbl/user_parin.f90
...
```

2. Set an additional path in the configuration file `.mrun.config` to allow `mrun` to find and include this file:

```
%add_source_path    $base_directory/USER_CODE/$fname
```

3. Modify the interface routines according to your needs.

Using the User-Interface with mrun

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

1. Copy the default (empty) user-interface files that you need (e.g. user_module.f90, user_parin.f90, user_actions.f90) to a directory of your choice, e.g.:

```
cd ~/palm/current_version
mkdir -p USER_CODE/example_cbl
cp trunk/SOURCE/user_module.f90 USER_CODE/example_cbl/user_module.f90
cp trunk/SOURCE/user_parin.f90  USER_CODE/example_cbl/user_parin.f90
...
```

2. Set an additional path in the configuration file `.mrun.config` to allow `mrun` to find and include this file:

```
%add_source_path    $base_directory/USER_CODE/$fname
```

3. Modify the interface routines according to your needs.
4. Start a PALM run by executing

```
mrun -d 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.

Using the User-Interface with mrun

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

1. Copy the default (empty) user-interface files that you need (e.g. user_module.f90, user_parin.f90, user_actions.f90) to a directory of your choice, e.g.:

```
cd ~/palm/current_version
mkdir -p USER_CODE/example_cbl
cp trunk/SOURCE/user_module.f90 USER_CODE/example_cbl/user_module.f90
cp trunk/SOURCE/user_parin.f90  USER_CODE/example_cbl/user_parin.f90
...
```

2. Set an additional path in the configuration file `.mrun.config` to allow `mrun` to find and include this file:

```
%add_source_path    $base_directory/USER_CODE/$fname
```

3. Modify the interface routines according to your needs.
4. Start a PALM run by executing

```
mrun -d 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.

- **The modified user-interface file cannot be pre-compiled by using** `mbuild`!

Using the User-Interface with mrun

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

1. Copy the default (empty) user-interface files that you need (e.g. user_module.f90, user_parin.f90, user_actions.f90) to a directory of your choice, e.g.:

```
cd ~/palm/current_version
mkdir -p USER_CODE/example_cbl
cp trunk/SOURCE/user_module.f90 USER_CODE/example_cbl/user_module.f90
cp trunk/SOURCE/user_parin.f90  USER_CODE/example_cbl/user_parin.f90
...
```

2. Set an additional path in the configuration file `.mrun.config` to allow `mrun` to find and include this file:

```
%add_source_path    $base_directory/USER_CODE/$fname
```

3. Modify the interface routines according to your needs.
4. Start a PALM run by executing

```
mrun -d 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.

- **The modified user-interface file cannot be pre-compiled by using** `mbuild`!
- The above method allows to use different user-interfaces for different runs. Just store the respective interface-files in subdirectories `USER_CODE/abcd`, `USER_CODE/cdef`, etc. and start `mrun` with option “`-d abcd`“, “`-d cdef`“, etc.