Multi-Model-Driver (MMD) User Manual

Version 1.1

Astrid Kerkweg[1],
& Patrick Jöckel[2]

[1] Institute for Atmospheric Physics
University of Mainz
55099 Mainz, Germany
kerkweg@uni-mainz.de
[2] Deutsches Zentrum für Luft-und Raumfahrt (DLR),
Institut für Physik der Atmosphäre,
Oberpfaffenhofen, D-82234 Weßling, Germany
patrick.joeckel@dlr.de

This manual is available as electronic supplement of our article "The 1-way on-line coupled atmospheric chemistry model system MECO(n): Part II: On-line Coupling " in Geosci. Model Dev. (2011), available at: http://www.geosci-model-dev.net

Date: November 11, 2011

Contents

1 Introduction

This manual is one part of a detailed description of the on-line coupling via the Multi-Model-Driver (MMD). MMD couples models following a client-server approach. It consists of three parts:

- The MMD library managing the data exchange between the different executables/models,
- the server MESSy submodel MMDSERV, providing the coarse grid data required by the client and
- the client MESSy submodel MMDCLNT, requesting the input data from the server and subsequently interpolating these data for use in the model.

The MMD library is described in the MMD library manual, which is part of the same electronic supplement this manual is published in. This manual, in contrast, is dedicated to the MMD MESSy submodels MMDCLNT and MMDSERV. In the current implementation, the ECHAM5/MESSy general circulation model is supported as server model, and the limited-area model COSMO/MESSy as server and/or client model. While the coupling layout, (i.e., which and how many model instances are run concurrently and which instance gets how many and which process entities (PEs)) is determined within the MMD library, the client submodel MMDCLNT drives the coupling between the two models of a client-server pair. Within the MMDCLNT namelist file the coupling frequency, i.e., how often data is exchanged between the server and the client, and the *exchange fields*[1] are specified.

After the data exchange MMDCLNT interpolates the coarse grid data to the COSMO model grid using its submodel INT2COSMO. INT2COSMO is based on INT2LM as provided by the German Weather Service (DWD) for the interpolation of the initial and boundary data for the COSMO model. INT2COSMO and INT2LM contain basically the same code, but for distinction we hereafter refer to the MMDCLNT submodel as INT2COSMO, i.e., the on-line preprocessing tool, and to INT2LM as the standard off-line (or stand-alone) application. The basic tasks of the server are to impose its time and date setting (except the time step length) on the client and to provide the requested data to the client.

In the first part of the manual those files which have to be modified by a user of the system are explained: Sect. 2 illustrates the run-script settings, Sect. 3 goes into detail about the individual namelist entries in the MMDCLNT namelist file and, for an overview, Sect. 4 roughly summarises the coupling work flow.

The second part of the manual is for those users, which want to expand the coupling or simply want to know more details about the coupling implementation: in Sects. 5 and 6 these are described for the client and the server, respectively. The superposition of the parallel decomposed COSMO model and INT2COSMO grids poses a specific challenge, which solution is discussed in Sect. 5.4.

Last but not least, the code changes of the individual model codes, which are required in order to enable the on-line coupling, are listed for the different code sources, i.e., INT2COSMO, the COSMO model and the ECHAM5 model, in the Sections 7, 8 and 9, respectively.

[1]The Appendix contains a glossary explaining some terms repeatedly used here. The terms from the glossary are written in italics throughout the article.

2 The run-scrip `xmessy_mmd`

The run-script consists of three major sections:

1. The first section contains queueing system settings. All queueing systems MESSy was sofar used with are listed in this run-script. New queueing systems can be added easily. The user has to activated the appropriate setup for the computing system he/she is using.
2. The second section is the one which needs specifications according to the actual model simulation. Thus this section is subject to changes by every user for a specific simulation.
3. The last section contains all MESSy and machine specific settings. These should not be changed by a model user. Only, if a new system is added, changes are also required here.

Here, we focus on the second block of the run-script, i.e., that part that needs to be adapted by the user for a specific model setup. Its start is indicated in the run-script by the comment:

```
###############################################################################
### USER DEFINED GLOBAL SETTINGS
###############################################################################
```

The settings are described one after the other as they are aligned in the script:

- `EXP_NAME`: This is the name of the experiment. This variable is copied to the CHANNEL[2] namelist. All CHANNEL output files will start with this experiment name. Its maximal length is 14 characters.
- `WORKDIR`: This is the directory in which the simulation is actually performed. In this directory subdirectories are created by the run-script and all data written during the simulation are placed into these (sub-)directories. For each model instance (see below) a subdirectory named by the instance number is created. Note: for most scheduling systems the log-file will be placed in the directory from which the run-script is submitted.
- `START_YEAR`, `START_MONTH`, `START_DAY`, `START_HOUR`: These are the start date/time components. They are copied to the TIMER[3] namelist defining the start date/time of the simulation. Additionally, they are copied to all namelists which require start time dependent entries. For instance, for an emission file containing monthly averaged emission fluxes the knowledge of the month in which the simulation starts is required. Nudging is another important example depending on the start date components.
- `STOP_YEAR`, `STOP_MONTH`, `STOP_DAY`, `STOP_HOUR`: These date components are copied to the TIMER and nudging namelists to determine the end of a simulation.
- `NML_SETUP`: This variable determines which namelist setup is used. In the subdirectory `messy/nml/` within the MESSy distribution different namelist setups are available. `NML_SETUP` selects the name of the subdirectory, which should be used. If an ECHAM5/MESSy-only simulation is performed, the subdirectory contains only the namelists required for an ECHAM5/MESSy simulation. For the coupled simulations the respective directory contains as many subdirectories as coupled instances exist. The numbers in the coupling layout (see below) are the same as the numbers of the namelist subdirectories.

[2]The CHANNEL submodel is described in detail in the electronic supplement of Jöckel et al. (2010).
[3]The TIMER submodel is described in detail in the electronic supplement of Jöckel et al. (2010).

- `OFT` (Output File Type): At the time being it can be chosen between netCDF[4] and parallel-netCDF[5], if the latter is available. This flag is copied to the CHANNEL namelist.

- `QWCH`: This should be set to the available wall-clock hours in a queue and is copied to the QTIMER[6] namelist.

- `INSTANCE`: This gives the number and type of model instances running simultaneously in the MPI environment. The letters "E" and "C" indicate whether in this instance an ECHAM5/MESSy model or a COSMO/MESSy model is executed.

```
### ========================================================================
### SELECT MODEL INSTANCES E=ECHAM5 (always first, if used), C=COSMO,
###                        M=MPIOM
###                        other = MBM
### ========================================================================
INSTANCE[1]=E
INSTANCE[2]=C
INSTANCE[3]=C
```

  If ECHAM5/MESSy is the coarsest server (*master server*), this needs to be the **first** instance. If ECHAM5/MESSy or COSMO/MESSy are run alone, only one instance is set. The run-script can also be used to run other MESSy models, e.g., the MESSy basemodel BLANK (`INSTANCE[1]=blank`), CAABA (`INSTANCE[1]=caaba`) or MPIOM (`INSTANCE[1]=M`) can be run as autonomous model with the same run-script.

- `MMDSERVID`: For each model instance the server of the model needs to be determined. The server of a model is defined by its instance number.

```
### ========================================================================
### SET MMD SERVER IDs (-1: MASTER SERVER)
### ========================================================================
MMDSERVID[1]=-1
MMDSERVID[2]=1
MMDSERVID[3]=2
```

  The *master server* is indicated by "-1" because the *master server* has no server itself. For the above example, the server of instance number 2 is the first instance, i.e., the ECHAM5/MESSy model. The second model itself is server to the third instance. `MMDSERVID[3]=1` would imply that the third model also gets its data directly from ECHAM5/MESSy.

- `NPX`, `NPY` and `NPL`: `NPX` and `NPY` determine the parallel domain decomposition of the processes in x and y direction, respectively. For the COSMO model these are copied to `nprocx` and `nprocy` in the COSMO namelist `&RUNCTL` in the `INPUT_ORG.nml` namelist file. For ECHAM5/MESSy these entries are copied to `NPROCA` and `NPROCB` in the `&RUNCTL` namelist of the `ECHAM5.nml` namelist file. `NPL` is only of importance for ECHAM5/MESSy, as the vector length `NPROMA` is set to `NPL`.

- ECHAM5 specific settings:

[4] *http://www.unidata.ucar.edu/software/netcdf/*
[5] *http://www.mcs.anl.gov/parallel-netcdf*
[6] The QTIMER submodel is described in Jöckel et al. (2010).

  - ECHAM5_HRES,ECHAM5_VRES: Spectral and vertical resolution of ECHAM5. ECHAM5_HRES is one of (T106, T85, T63, T42, T31, T21, T10), whereas ECHAM5_VRES is one of (L19, L31ECMWF, L41DLR, L39MA, L90MA). Note that ECHAM5 always requires input data which matches the chosen resolution.
  - MPIOM_HRES, MPIOM_VRES: horizontal and vertical resolution of MPIOM, when it is chosen as a submodel. (Note: this is not yet available.) MPIOM_HRES is one out of (GR60, GR30, Gr15, TP04, TP40) and MPIOM_VRES one of (L3, L20, L40).
  - ECHAM5_NUDGING: This LOGICAL is set to T, if nudging of the ECHAM5 model is requested. The nudging coefficients in the ECHAM5 namelist file (namelist &NDGCTL) must be set accordingly.
  - ECHAM5_LAMIP: Switch on sea-surface temperature (sst) and sea-ice forcing via AMIP-like data for ECHAM5.
  - NML_ECHAM: name of the ECHAM5 namelist file. As the ECHAM5 namelists include some resolution dependent entries, it is convenient to work with resolution dependent ECHAM5 namelist files.

- COSMO_SUBDIR, COSMO_EXTNAME and COSMO_EXTGRID: These entries are used to determine the INT2COSMO namelist entries required to access the external data file. COSMO_EXTNAME fills ylmext_lfn and COSMO_EXTGRID contains the sizes of the external data file, which naturally depend on the external data file and are required as individual entries in the INT2COSMO namelist &DATA. ylmext_cat is filled by COSMO_EXTDIR, which is composed of a general data input path (INPUTDIR_COSMO_EXT) and a subdirectory (COSMO_SUBDIR) in this input path. COSMO_SUBDIR has to be defined individually for each instance, while a default value (as part of the standard MESSy input directory) exists for INPUTDIR_COSMO_EXT. However, INPUTDIR_COSMO_EXT can be user-defined for each individual instance.

  ```
  ### -> COSMO_EXTDIR[.] = ${INPUTDIR_COSMO_EXT[.]}/$COSMO_SUBDIR[.]
  COSMO_SUBDIR[1]=
  COSMO_EXTNAME[1]=
  COSMO_EXTGRID[1]=

  COSMO_SUBDIR[2]=climatology
  COSMO_EXTNAME[2]=europe.nc
  COSMO_EXTGRID[2]="ie_ext=101, je_ext=107,"

  COSMO_SUBDIR[3]=external
  COSMO_EXTNAME[3]=lm_d1_g0.165_463x383
  COSMO_EXTGRID[3]="ie_ext=463, je_ext=383,"
  ```

  The number in brackets is the instance number. In the example with ECHAM5/MESSy as first instance the block with instance number 1 must be empty.

The above listed variables need to be set. Here, additional variables are listed, which can be set, if the default settings should not be used:

- BASEDIR: This is the directory of the model distribution.
- DATABASEDIR: Base directory for model input data.

- `INPUTDIR_ECHAM5_INI`: Directory containing the input data for the ECHAM5 model.
- `INPUTDIR_ECHAM5_SPEC`: Directory containing the `*_spec` and `*_surf` files for the ECHAM5 initialisation. These depend on the resolution and start date. With `INPUTDIR_ECHAM5_SPEC`, the user can put the initial files specific for the start date of his/her simulation into a private directory and use the default directory for the others.
- `INPUTDIR_AMIP`: Directory of the sst and sea-ice data for ECHAM5.
- `INPUTDIR_NUDGE`: Directory of the nudging data files for ECHAM5.
- `FNAME_NUDGE`: Name of the ECHAM5 nudging data files. This differs dependent on the source of the data. (ERA40 or analysis (ANALY))
- `INPUTDIR_MPIOM`: Directory containing MPIOM input data.
- `INPUTDIR_COSMO`: Directory containing COSMO input data.
- `INPUTDIR_MESSY`: Directory containing input data for the MESSy submodels.
- `USE_PREREGRID_MESSY`: It is possible to provide the MESSy input data on the specific horizontal Gaussian grid, i.e., in pre-regridded form. This is used when `USE_PREREGRID_MESSY = T`. Note: this only works for ECHAM5/MESSy, as the regridder only works on rectangular grids.
- SPECIAL MODES:
  - `MEASUREMODE`: Measure memory use. This is only available on specific machines.
  - `TESTMODE`: Test mode of the run-script, exits before starting the executable.
  - `TPROFMODE` and `TPROFCMD`: A special mode, for performance monitoring, which is only available on IBM and on maximum 1 node.

The user specified block ends with the marker:

```
###############################################################################
###############################################################################
### =========================================================================
###############################################################################
### DO NOT CHANGE ANYTHING BELOW THIS LINE !!!
###############################################################################
### =========================================================================
###############################################################################
###############################################################################
```

3 The MMDCLNT namelists

The namelist file `mmdclnt.nml` contains two parts:

I) a server independent part (the `&CPL`-namelist) and

II) the server dependent namelists `&CPL_ECHAM` and `&CPL_COSMO`.

```
! -*- f90 -*-
&CPL
CPL_IOEVENT      = 10,'minutes','first',0
READEXT_IOEVENT  = 1,'years','none',0
/

&CPL_ECHAM
! ############################################################################
!
! ############################################################################
! ### MANDATORY FIELDS
! ############################################################################
!***************************************************************************
FIELD(1)   = 'g3b','aps', 'COSMO_ORI', 'PS', '', F, F, F , ''
!***************************************************************************
FIELD(2)   = 'ec2cosmo','T_S', 'COSMO_ORI','T_S', '', T, T, F, ''
!***************************************************************************
FIELD(3)   = 'g3b','slf', 'COSMO_ORI','FR_LAND', '', T, F, F, ''
!***************************************************************************
FIELD(4)   = 'g1a','tm1', 'COSMO_ORI','T', '', T, T, F, ''
!***************************************************************************
FIELD(5)   = 'g1a','qm1', 'COSMO_ORI','QV', '', T, T, F, ''
!***************************************************************************
FIELD(6)   = 'g1a','xlm1', 'COSMO_ORI','QC', '', T, T, F, ''
!***************************************************************************
FIELD(7)   = 'g1a','xim1', 'COSMO_ORI','QI', '', T, T, F, ''
!***************************************************************************
FIELD(8)   = 'g2a','um1', 'COSMO_ORI','U', '', T, T, F, ''
!***************************************************************************
FIELD(9)   = 'g2a','vm1', 'COSMO_ORI','V', '', T, T, F, ''
!***************************************************************************
FIELD(10)  = 'g3b','geosp', '#XXX','FIS', '', F, F, F, ''
!***************************************************************************
FIELD(11)  = 'g3b','wl', 'COSMO_ORI','W_I', '', T, F, F, ''
!***************************************************************************
FIELD(12)  = 'g3b','sni', 'COSMO_ORI','W_SNOW', '', T, T, F, ''
!***************************************************************************
FIELD(13)  = 'g3b','tsi', 'COSMO_ORI','T_SNOW', '', T, T, F, ''
!***************************************************************************
FIELD(14)  = 'ec2cosmo','W_SO_REL', 'COSMO_ORI','W_SO', '', T, F, F, ''
! ############################################################################
! ### OPTIONAL FIELDS
! ############################################################################
FIELD(20)  = 'Test','Test_Ar', 'mmdclnt','Test_Ar', '', F, F, F, ''
!***************************************************************************
FIELD(21)  = 'tracer_gp_m1','O3', 'tracer_gp','O3', 'QFTV', T, T, F, ''
!***************************************************************************
FIELD(22)  = 'ptrac_gp','wetradius', 'ptrac_gp','wetradius', 'QTFV', T, F, F, ''
!***************************************************************************
FIELD(23)  = 'jval_gp','J_O1D', 'mmdclnt','J_O1D', 'QFTV', F, F, T, 'GP_3D_MID'
!***************************************************************************
FIELD(24)  = 'import_rgt','RGTO012_CO','mmdclnt','RGTO012_CO','M',F,F,T,'#UNKNOWN'
! ############################################################################
/
```

Figure 1: Example namelist file of MMDCLNT (`mmdclnt.nml`). Part I: CPL-namelist and CPL-ECHAM namelist.

```
&CPL_COSMO
! ############################################################################
!
! ############################################################################
! ### MANDATORY FIELDS
! ############################################################################
FIELD(1)  = 'COSMO','ps', 'COSMO_ORI','PS', '', F, F, F, ''
!***************************************************************************
FIELD(2)  = 'COSMO','t_s', 'COSMO_ORI','T_S', '', T, T, F, ''
!***************************************************************************
FIELD(3)  = 'COSMO_ORI','FR_LAND', 'COSMO_ORI','FR_LAND', '', T, F, F, ''
!***************************************************************************
FIELD(4)  = 'COSMO','tm1','COSMO_ORI','T', '', T, T, F, ''
!***************************************************************************
FIELD(5)  = 'COSMO','qv','COSMO_ORI','QV', '', T, T, F, ''
!***************************************************************************
FIELD(6)  = 'COSMO','qc','COSMO_ORI','QC',  '', T, T, F, ''
!***************************************************************************
FIELD(7)  = 'COSMO','qi','COSMO_ORI','QI',  '', T, T, F, ''
!***************************************************************************
FIELD(8)  = 'COSMO','um1','COSMO_ORI','U',  '', T, T, F, ''
!***************************************************************************
FIELD(9)  =  'COSMO','vm1', 'COSMO_ORI','V','', T, T, F, ''
!***************************************************************************
FIELD(10) = 'COSMO','t_so', 'COSMO_ORI','T_SO', '', T, F, F, ''
!***************************************************************************
FIELD(11) = 'COSMO','w_so','COSMO_ORI','W_SO',  '', T, F, F, ''
!***************************************************************************
FIELD(12) = 'COSMO','t_snow', 'COSMO_ORI','T_SNOW', '', T, T, F, ''
!***************************************************************************
FIELD(13) = 'COSMO','w_snow', 'COSMO_ORI','W_SNOW', '', T, T, F, ''
!***************************************************************************
FIELD(14) = 'COSMO','w_i', 'COSMO_ORI','W_I', '', T, F, F, ''
!***************************************************************************
FIELD(15) = 'COSMO','qv_s', 'COSMO_ORI','QV_S', '', T, T, F, ''
!***************************************************************************
FIELD(16) = 'COSMO_ORI','FRESHSNW', 'COSMO_ORI','FRESHSNW', '', T, F, F, ''
!***************************************************************************
FIELD(17) = 'COSMO_ORI','HSURF', 'COSMO_ORI','HSURF', '', T, F, F, ''
!***************************************************************************
FIELD(18) = 'COSMO','ppm1', 'COSMO_ORI','PP', '', T, T, F, ''
!***************************************************************************
FIELD(19) = 'COSMO_ORI','SOILTYP', 'COSMO_ORI','SOILTYP', '', T, F, F, ''
!*************************************************************************
! ############################################################################
! ### OPTIONAL FIELDS
! ############################################################################
FIELD(20) = 'Test','Test_Ar', 'mmdclnt','Test_Ar', '', F, F, F, ''
!***************************************************************************
!***************************************************************************
FIELD(21) = 'tracer_gp_m1','all', 'tracer_gp','*', 'QFTV', T, T, F, ''
!***************************************************************************
! ############################################################################
/
```

Figure 2: Example namelist file of MMDCLNT (`mmdclnt.nml`). Part II: CPL_COSMO namelist.

I) **The `&CPL`-namelist:**
The `&CPL`-namelist (Fig. 1) contains two *events*[7]:

- the first (`CPL_IOEVENT`) determines the coupling frequency to the server model, i.e., how often data is exchanged between server and client model,
- the second *event* (`READEXT_IOEVENT`) drives the update of the external data required by `INT2COSMO`.

For the definition of the coupling *event* the user has to be aware of three limitations:

- To simplify the data exchange between the coupled models, the coupling interval is internally converted to seconds. As this conversion is not well defined for the units `'months'` and `'years'`, the coupling interval must be specified in `'steps'`, `'seconds'`, `'minutes'`, `'hours'` or `'days'`.
- As for all other *events*, the user has to define a multiple of the model time step length, otherwise the simulation is terminated.
- The user has to take care that the coupling interval is a multiple of the time steps of the client and the server model.

II) **The `&CPL_ECHAM`/`&CPL_COSMO`-namelist:**
The second part of the namelist file contains a list of *exchange fields* required to fully initialise and drive the client COSMO/MESSy model. The *exchange fields* are unambiguously identified by their *channel* and *channel object* names. As these usually differ between ECHAM5/MESSy and COSMO/MESSy, the namelists depend on the server.

Figure 1 shows a typical `&CPL_ECHAM` namelist for the coupling to ECHAM5/MESSy and Fig. 2 shows the namelist `&CPL_COSMO` used for the coupling to a COSMO/MESSy model as server. The structure of the two namelists is identical. Each *exchange field* is defined by one namelist entry of TYPE `FIELD`:

```
FIELD(.) = 'SERV_CHANNEL', 'SERV_OBJECT', 'CLNT_CHANNEL', 'CLNT_OBJECT'
         , 'INTERPOL_METHOD', L_INITIAL, L_BOUND, L_INPUT, 'CLNT_REPR'
```

`FIELD` is a variable of TYPE `T_EXCH_IO`:

```
  TYPE CHAOBJ_NAMES
     CHARACTER(LEN=STRLEN_CHANNEL)  :: CHA = '' ! CHANNEL NAME
     CHARACTER(LEN=STRLEN_OBJECT)   :: OBJ = '' ! OBJECT  NAME
  END TYPE CHAOBJ_NAMES

  TYPE T_EXCH_IO
     TYPE(CHAOBJ_NAMES) :: SERVER
     TYPE(CHAOBJ_NAMES) :: CLIENT
     CHARACTER(LEN=4)   :: C_INTERPOL = ''      ! INTERPOLATION METHOD
     ! Specify target field
     LOGICAL            :: L_INITIAL  = .FALSE. ! INITIAL FIELD
     LOGICAL            :: L_BOUND    = .FALSE. ! BOUNDARY FIELD
```

[7]The generic submodel TIMER and the definition and functionality of *events* are described in the manual about TIMER within the electronic supplement of Jöckel et al. (2010).

```
   LOGICAL             :: L_INPUT    = .FALSE. ! INPUT FIELD
   CHARACTER(LEN=STRLEN_MEDIUM) :: C_REPR =’’ ! REPRESENTATION STRING
END TYPE T_EXCH_IO

! MAXIMAL NUMBER OF EXCHANGE FIELDS
INTEGER, PARAMETER                              :: NMAX_EXCH = 1000
TYPE(T_EXCH_IO), DIMENSION(NMAX_EXCH), SAVE :: FIELD
```

- The first two structure components of TYPE CHARACTER specify the *channel* and *channel object* name of the *exchange field* on the server side. For instance, the surface pressure field in ECHAM5/MESSy is defined in the *channel* 'g3b' with the *channel object* name 'aps' (see `FIELD(1)` in Fig. 1).
- The third and fourth structure components of TYPE CHARACTER name the *channel* and *channel object* of the *exchange field* in the client model. For `FIELD(1)` in Fig. 1 this is the *channel* 'COSMO_ORI' and the *channel object* 'PS'.

  For the client *channel object* names wildcards are allowed. A `'*'` replaces an arbitrary number of characters or digits, whereas `'?'` replaces exactly one character or digit. Based on wildcards, it is possible to address a number of *channel objects* of one *channel* with one namelist entry. The only restriction for wildcard usage is that the name of the *channel object* on the server side must be identical to that on the client side[8], because the names for the server *channel object*s are overwritten by the client *channel object* names, if wildcards are used. For instance, the entire tracer set can be coupled by setting `'CLNT_CHANNEL', 'CLNT_OBJECT'` to `'tracer_gp', '*'` or all photolysis rates are coupled by `'jval_gp', 'J_*'`. Due to the initialisation of prognostic variables at the beginning of each time step in COSMO/MESSy in the subroutine `initialize_loop` in `lmorg.f90` the server *channel* for the coupling of the tracers needs to be `'tracer_gp_m1'`. In case of ECHAM5 the fields in `'tracer_gp_m1'` and `'tracer_gp'` are identical at the beginning of the time loop.
- The fifth structure component is a CHARACTER of length 4. It determines the interpolation method. This is only required for the *additional fields*, as for the *INT2COSMO inherent fields* the interpolation method is determined inside of INT2COSMO. Possible interpolation methods are: 'Q' for quadratic; 'L' for linear and 'M' for match interpolation. Thus the first character must be set to one of 'Q', 'L', or 'M'. The second and the third character demand monotonicity and positive definiteness, respectively, if set to T. The default value, however, is F. If the fourth character is V, the field will be interpolated also in the vertical direction. However, this is only possible for 3-D-or 4-D-fields of which the number of vertical levels equals the number of vertical levels in the model. For instance, the fifth string of FIELD(21) in `&CPL_ECHAM` determines that the ozone tracer is interpolated horizontally by quadratic interpolation and in addition vertically. No care is taken to ensure monotonicity, but positive definiteness is requested.
- The three logicals indicate the data destination (*initial*, *boundary* or *input*) of the interpolated field. *Mandatory fields* can be *initial* and *boundary fields*. For the *mandatory fields* the entries for the data destination types in the namelist can be omitted, as they are set according to the COSMO variables `yvarini` and `yvarbd`. These variables list the *initial* and *boundary fields* required for the chosen COSMO setup. If *initial* or *boundary fields* are required according to `yvarini` or `yvarbd` and the data destination flags are not set `.TRUE.`

[8]This is usually not the case for the basemodels ECHAM5 and COSMO, but for the MESSy submodels.

in the namelist, the namelist settings are ignored. If a field destination is requested (in addition to `yvarini` or `yvarbd`) as *initial* or *boundary field*, however, this request is not overwritten.

For the *optional fields* the choice of *initial* and/or *boundary* and of *input* destination is exclusive, as *input* already implies *initial* and the provision of *boundary* data is meaningless, since the field is overwritten each coupling time step. For instance, for the prognostic variables water vapour and cloud water (`FIELD(5)` and `FIELD(6)` in Fig. 1) the calculation of the *initial* and *boundary fields* is requested, whereas for the land fraction (`FIELD(3)` in Fig. 1) only the *initial field* is calculated. As tracers are prognostic variables, *initial* and *boundary fields* are requested for Ozone (`FIELD(21)`). In contrast, the fields `FIELD(23)` and `FIELD(24)` are *input fields*.

– The last component of the variable `FIELD` contains the client *representation*[9]. It is only required for *additional fields*, for which `L_INPUT` is `.TRUE.`. In this case the memory for a field is neither defined by a MESSy submodel nor by the basemodel itself and consequently, MMDCLNT has to define the respective *channel object* itself, which is indicated by giving `'mmdclnt'` as client *channel* name in the third `FIELD` entry in the MMDCLNT namelist file. For these fields the *representation* must be known as MMDCLNT needs to allocate the memory for the respective field itself.

For instance, in `FIELD(23)` in the `&CPL_ECHAM` namelist, the photolysis rate of O^1D from the ECHAM5/MESSy submodel JVAL (*channel* name `'jval_gp'`, *channel object* name `'J_O1D'`) is defined as *input field* of the regional model. If JVAL is not switched on in COSMO/MESSy, MMDCLNT needs to define the *channel object* itself. The photolysis rates are defined at the center of the grid boxes. Thus the *representation* of a photolysis rate is a priori known and the *representation* name for the client can be specified (here, `'GP_3D_MID'`).

In cases where the *representation* is not a priori known, it is deduced from the *representation* of the server *channel object*. This heuristic procedure, triggered by the entry `'#UNKNOWN'` (see `FIELD(24)` in Fig. 1), is described in detail in Sect. 5.1.2.

In addition to the coupling of standard 2-D and 3-D data fields, the coupling of 4-D data fields is implemented. They are treated exactly in the same way. However, due to differences in the implementation of tracers (Jöckel et al., 2008) and the implementation of prognostic variables in the COSMO model, it is not possible to couple the 4-D tracer field directly. Nevertheless, each individual tracer can be coupled, as the individual tracers are accessible as 3-D *channel objects* (e.g., `FIELD(21)` in Fig. 1). To simplify the handling of large tracer sets, wildcards can be used for the client *channel object* names in the namelist: `'*'` replaces an arbitrary number of characters, `'?'` replaces exactly one character. For instance, `FIELD(25)` would request all tracers available in the *channel* `'tracer_gp_m1'`. Of course, wildcards in the *channel object* names can be used for other *channels* as well.

4 Basic coupling setup

The diagram in Fig. 3 sketches the sequence of operations in an on-line coupled simulation.

- **MMD setup**: Before any submodel specific initialisation takes place, the Multi-Model-Driver (MMD) is set up. The MMD library routines setting up the message passing interface (MPI)

[9]For a description of *representations* see the CHANNEL manual, which is part of the electronic supplement of Jöckel et al. (2010).

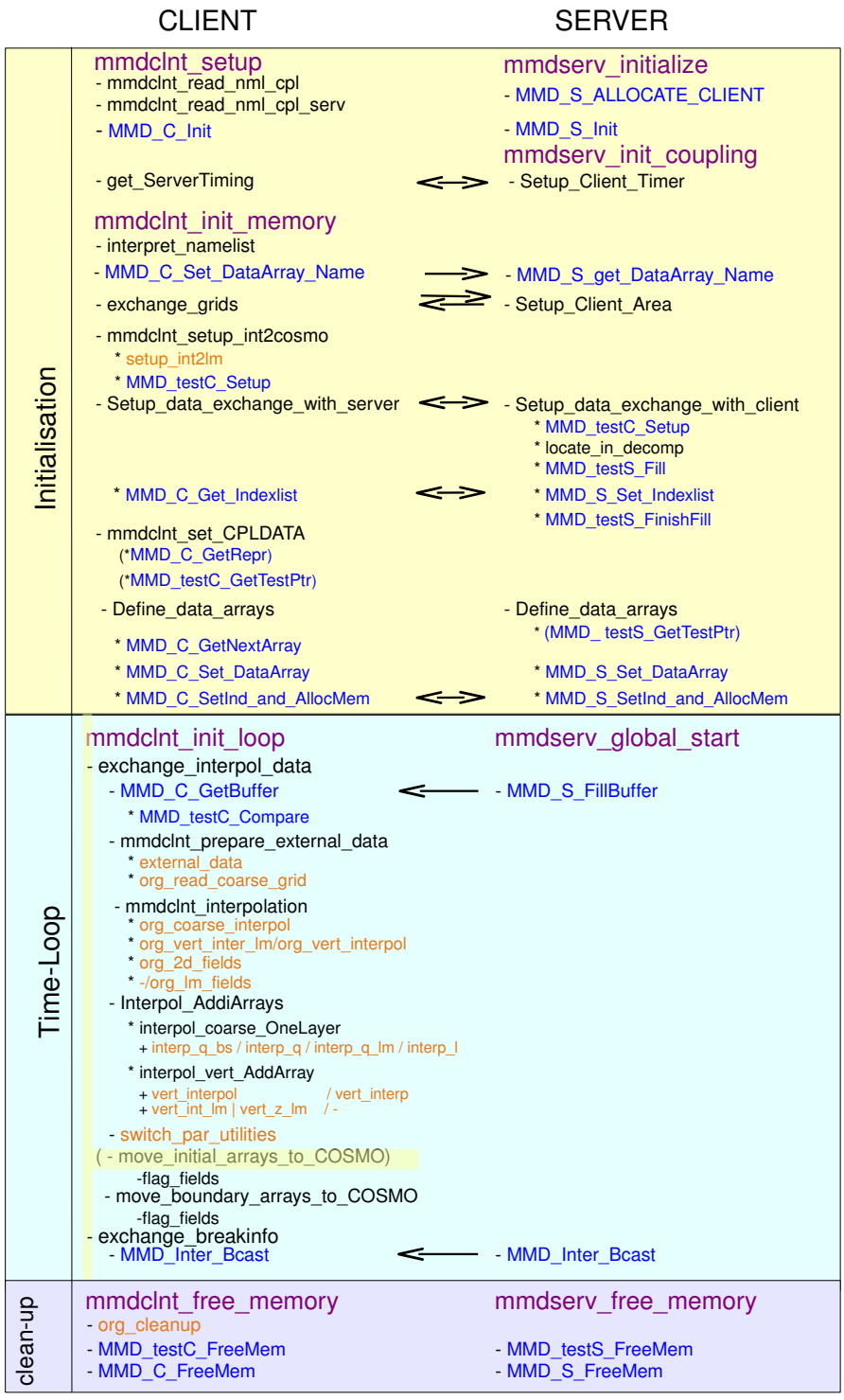

Figure 3: Call sequence of the on-line coupling routines in server and client in ECHAM5/MESSy ($\rightarrow$ COSMO/MESSy)^n: The background colours indicate, whether the routines are called during the initialisation (yellow), the time loop (cyan) or in the finalisation phase (lilac). The routine names are also colour-coded: the MESSy entry points directly called by `messy_main_control` are written in lilac. The MMD library routines are indicated in blue and the original INT2LM routines in orange. Arrows depict the data exchange between client and server.

environment are called from the basemodel. The determination of the model topology and the communicator definition are explained in an extra manual about the MMD library[10], as these routines work inside the MMD library. As model topology, we understand the layout of all server and client dependencies and the distribution of the models on the available number of process entities (PEs) or MPI tasks.

The topology is determined by the MMD library namelist `MMD_layout.nml`, which is written by the run-script `xmessy_mmd` as determined by the user. The MMD library namelist file `MMD_layout.nml` is read, broadcasted and interpreted within the MMD library. All communicators for intra- and inter-model communication are determined in accordance to the model topology.

- **Synchronisation**: To ensure that all models start, *restart* and stop at the same date and time, the date and time settings of all coupled models need to be synchronised. This is achieved if one model determines the timing of all models. If in each server-client pair the server dictates the timing, in the end the *master server* determines the timing of all models. Consequently, the TIMER namelist of the *master server* determines the timing of all client models. However, it is important to note, that each model uses its own time step length, as only the dates are synchronised.

- **Data exchange initialisation**:

  - In contrast to the timing, the data exchange during the coupling process is completely controlled by the client model:
    * The client requests data for its specific model domain from the server,
    * the client namelists `&CPL_ECHAM` or `&CPL_COSMO` determine which data fields are exchanged and
    * the client namelist `&CPL` determines the frequency of the data exchange.

    These requests are processed by the server during the initialisation phase.
  - The client and the server acquire POINTERs to the data fields required for the data exchange and (on the client side) for the interpolation.
  - Additionally, the buffers for the data exchange are allocated within the MMD library.

- **The data exchange**: During the time loop the *exchange fields* are made available by the server submodel MMDSERV (subroutine `MMD_S_FillBuffer`). These fields are copied by the client and interpolated according to the namelist settings.

- **Finalisation phase**: At the end of the integration coupling specific memory is deallocated.

5 The Client

All information required during the coupling process is contained within the variable `CPLDATA`, which TYPE is a Fortran95 structure (`T_COUPLE_DATA`) and which is allocated to the actual number of *coupling fields*.

[10]The MMD library manual is part of the same electronic supplement as this manual.

```
TYPE PTR_4D_ARRAY
   REAL(DP), DIMENSION(:,:,:,:), POINTER :: PTR
END TYPE PTR_4D_ARRAY

TYPE CHAOBJ_NAMES
   CHARACTER(LEN=STRLEN_CHANNEL)    :: CHA = '' ! CHANNEL NAME
   CHARACTER(LEN=STRLEN_OBJECT)     :: OBJ = '' ! OBJECT  NAME
END TYPE CHAOBJ_NAMES

TYPE T_COUPLE_DATA
   ! CHANNEL AND CHANNEL OBJECT NAMES IN SERVER AND CLIENT
   TYPE(CHAOBJ_NAMES)                    :: SERVER
   TYPE(CHAOBJ_NAMES)                    :: CLIENT
   ! ORDER OF AXES IN REPRESENTATION ('X','Y','Z','N')
   CHARACTER(LEN=4)                      :: AXIS= ''
   ! DIMENSION LENGTH
   INTEGER, DIMENSION(4)                 :: ldimlen=0
   ! INTERPOLATION METHOD (only valid for arrays not included in vartab)
   ! 1.CHAR 'Q' quadratic; 'L': linear and 'M' match interpolation is possible
   ! 2.CHAR if 'T' positive definiteness  is required
   ! 3.CHAR if 'T' monotonicity           is required
   ! 4.CHAR if 'V' vertical interpolation is required
   CHARACTER(LEN=4)                      :: C_INTERPOL
   ! INPUT FIELD DELIVERED BY MMD
   REAL(DP), POINTER, DIMENSION(:,:,:,:) :: ptr_in
   ! INTERMEDIATE FIELD OF INT2COSMO
   REAL(DP), POINTER, DIMENSION(:,:,::,) :: ptr_i2c
   ! POINTER(S) TO COSMO/MESSy FIELD(S): DIMENSION == number of time levels
   TYPE(PTR_4D_ARRAY), DIMENSION(:), POINTER :: cosmo => NULL()
   ! POINTER TO COSMO/MESSy BOUNDARY FIELDs:
   !  (DIMENSION IS ALWAYS TWO FOR THE TWO BOUNDARY LAYER TIME LEVELS)
   TYPE(PTR_4D_ARRAY), POINTER, DIMENSION(:) :: cosmo_bd => NULL()
   ! RANK OF CLIENT FIELD (WITHOUT TIME LEVEL DIMENSION)
   INTEGER                               :: rank        = 0
   ! INDICATOR, IF FIELD IS IN VARTAB
   LOGICAL                               :: lvartab     = .FALSE.
   ! NAME OF VARIABLE IN VARTAB
   CHARACTER(LEN=10)                     :: vartab_name = ''
   ! INDEX OF FIELD in var_lm
   INTEGER                               :: vartab_idx   = 0
   ! INITIAL FIELDS REQUIRED ?
   LOGICAL                               :: L_INITIAL   = .FALSE.
   ! BOUNDARY FIELDS REQUIRED ?
   LOGICAL                               :: L_BOUND     = .FALSE.
   ! INPUT FIELD REQUIRED ?
   LOGICAL                               :: L_INPUT     = .FALSE.
   ! NAME OF REPRESENTATION
   CHARACTER(LEN=STRLEN_MEDIUM)          :: C_REPR      = ''
```

```
END TYPE T_COUPLE_DATA
```

This structure contains

- the *channel* and *channel object* names of the *exchange fields* for the server (`TYPE(CHAOBJ_NAMES) :: SERVER`) and
- the *channel* and *channel object* names for the client (`TYPE(OCHOBJ_NAMES) :: CLIENT`).
- information about the *dimensions* of the fields:
  - The *axis string* (`AXIS`) indicates the order of the 'X', 'Y', 'Z' and 'N' direction and
  - `ldimlen` contains the length of these four *dimensions*[11].
- `C_INTERPOL` is the flag specifying the interpolation method.
- The structure contains four POINTERs or *POINTER ARRAYs* for the access to the different data fields during the interpolation procedure and to the *target fields*. Depending on the source (*exchange field* or from external data) and the *target field* (i.e., *boundary* or *initial* and *input fields*) not all four POINTERs or *POINTER ARRAYs* are used for all *coupling fields*[12]:
  - The first POINTER (`ptr_in`) is used for the *in-fields*, i.e., the raw data sent from the server model.
  - The second POINTER (`ptr_i2c`) is associated to the *intermediate fields* generated by the horizontal (and vertical) interpolation within INT2COSMO.
  - The *POINTER ARRAY* `cosmo` is associated with the *target field* in the COSMO/MESSy model. For prognostic variables the dimension of the *POINTER ARRAY* is given by the number of time levels. Each of the POINTERs in the *POINTER ARRAY* is associated to one time level of the *target field*. For diagnostic variables the array dimension is always 1. For instance, the prognostic field for the temperature in COSMO is dimensioned by the three space dimensions and a time level dimension. Dependent on whether a two or three time level integration scheme is used, this fourth dimension is allocated by 2 or 3. All time levels have to be made accessible in MMDCLNT for the respective *target field*. Thus the *POINTER ARRAY* `cosmo` is also dimensioned according to the number of the time levels required by the integration scheme. This yields the POINTERs `CPLDATA(ii)%cosmo(nt)%ptr`, where `nt` is an index ranging from 1 to the number of time levels used in the COSMO model allowing to access the different time levels of the *target field*, i.e., `nt` is one of the time level indices `nnew`, `nnow` and `nold`, respectively. In contrast, a diagnostic variable does not depend on the integration scheme, thus one POINTER is sufficient to access a diagnostic *target field*.

[11] These are properties already defined and provided by the CHANNEL submodel. See the CHANNEL manual available in the electronic supplement of Jöckel et al. (2010) for further information.

[12] Note: the different meaning of the different fields:

- *exchange fields* are those fields exchanged with the server, they are not necessarily associated to a *target field*, as they might also be required for the interpolation and not as input for the client.
- *coupling fields* are all those fields contained in the variable `CPLDATA`, i.e., either *exchange fields* or fields additionally provided by INT2COSMO, e.g., calculated from the external data.
- *target field* can be an *input* or *initial field*, or the respective *boundary field* for prognostic variables.

The meaning of the individual fields is clarified within the remainder of the client description and in the glossary.

  - If boundary data is required for a field, the *POINTER ARRAY* `cosmo_bd` is allocated with 2 dimensions according to the two time levels required for the boundary data in the COSMO/MESSy model. Each of the POINTERs is associated to one level of the boundary data array.

To actually perform the coupling, additional information is required:

- The `rank` of the field,
- the information if a field is part of the variable table in INT2COSMO (LOGICAL `lvartab`), i.e., if the field is an *INT2COSMO inherent field*,
- the name in the variable table (`vartab_name`), if `lvartab = .TRUE.`, and
- the index of the variable in the variable table of INT2COSMO (`vartab_idx`), if `lvartab = .TRUE.`,
- the LOGICALs `L_INITIAL`, `L_BOUND` and `L_INPUT` indicating if initial, boundary or input data is required and
- the string `C_REPR`.

The meaning of these variables was already illustrated in the section about the namelist (Sect. 3) and will become clearer in the remainder of the client description.

5.1 Initialisation Phase

The main entry point for submodel initialisation in MESSy is `messy_initialize`. In contrast to this, MMDCLNT uses an even earlier entry point, i.e., `messy_setup` (compare Fig. 3). This is necessary, as the *master server* determines the timing of all models in the cascade. As the date and time setup is an essential part of a simulation, it is performed very early during the model setup, thus a very early entry point for MMDCLNT was required.

5.1.1 `mmdclnt_setup`

This subroutine performs the basic setup of MMDCLNT and the timing of the client.

- At the beginning two LOGICALs need to be set, which determine the information flow in the basemodel and the MESSy generic submodels:
  - The LOGICAL variable `L_IS_CLIENT` defined in `messy_main_data_bi` is set to `.TRUE.`. It is used in the COSMO model itself to switch off certain parts of the code dealing with the import of initial and boundary data (in `src_input.f90`).
  - `lforcedtime`, a second LOGICAL required for the synchronisation of the server and client model is also initialised to `.TRUE.`. In `timer_global_start` the LOGICAL for the trigger of the RERUN *event* `l_rerun` is only calculated, if `lforcedtime` is `.FALSE.`. If `l_rerun` were determined in the client TIMER itself, the client may finish unnoticed by the server and a dead lock in MPI would occur resulting in a model hang up.
- Subsequently, the MMDCLNT namelists are read:

  - First, the `&CPL`-namelist is read (in subroutine `mmdclnt_read_nml_cpl`) and the two `IO_TIME_EVENT`s `CPL_IOEVENT` and `READEXT_IOEVENT` are broadcasted.
  - Second, the server specific coupling namelist is read in the subroutine `mmdclnt_read_nml_cpl_serv`. Whether the ECHAM5/MESSy or the COSMO/MESSy specific namelist is read, is determined with the help of the MMD library function `MMD_C_GetServerType` (`mmd_client.f90`).

  After reading the respective namelist, the subroutine prints the read values into the log-file.

- In addition to the client submodel setup, the client part of the MMD library has to be initialised. This is done by the MMD library routine `MMD_C_Init`. Within this subroutine the C group communicators are determined and the number of PEs covered by the server model is retrieved. Parts of the MMD library internal information structure (`Me`), of which the dimensions depend on the number of server PEs are allocated within `MMD_C_Init`[13].

- **get_ServerTiming:** To get a meaningful simulation, the timings of the coupled models need to be synchronised. To achieve this, two important parts of information are exchanged between the client and the server:

  1.) The coupling interval determined in the client namelist is forwarded to the server:
  In the subroutine `get_ServerTiming` first the time interval of the coupling *event* `CPL_IOEVENT` is converted into seconds, as this is the unambiguous unit to exchange between the two models. The number of seconds equivalent to the coupling interval is sent to the server, which accordingly defines a coupling *event*. If the coupling interval is not a multiple of the server model time step length the simulation is terminated.

  2.) The server sends the complete date settings to the client in order to initialise the dates of the client basemodel:
  The `current_date`, the `resume_date`, the `start_date` and the `stop_date` are sent from the server to the client. The client re-initialises its date settings according to the server dates. However, two cases are distinguished:

    * In case of `lstart = .TRUE.` (for the client), the `start_date` is set to the `resume_date` of the server. This is done, as it is possible, that the server performs a *restart* while the client is started for the first time. In this case the `resume_date` of the server is the `start_date` of the client. Note that, if `lstart` is also `.TRUE.` for the server, the `resume_date` and the `start_date` are equal anyway.
    * In case of a *restart* of the client, the `current_date` is not set at all, as it is set by the TIMER later on anyway. More important, the client `resume_date` is calculated from the server `resume_date`. This is necessary, as the restart files for client and server are usually not output at the same time. They both stop after `l_rerun` was set `.TRUE.`, thus the client `resume_date` is behind the server `resume_date` by the difference of the server and the client time step lengths. Based on this, the client `resume_date` is calculated from the two model time step lengths and the server `resume_date`.

  The COSMO/MESSy time settings and counters are re-initialised within the subroutine `reinit_COSMO_time` provided by `messy_main_timer_bi`.

[13]The MMD library routines are described in detail in the MMD library manual, which is part of the same electronic supplement as the MMD user manual.

5.1.2 mmdclnt_init_memory

The second part of the initialisation takes place in `mmdclnt_init_memory`, as here the main memory allocation and field definitions are conducted.

- Due to technical reasons, the *events* themselves can not be initialised before the MESSy entry point `messy_init_memory`. Thus first of all, the TIMER *events* for the coupling (`CPL_EVENT`) and for reading the external data (`READEXT_EVENT`) are initialised. Additionally, the `BREAK_EVENT` is defined. This *event* is initialised to the length of the server time step. It is used to encounter every server time step, whether the simulation is interrupted at the end of the time step. This is inevitably necessary to ensure that the server and the client stop at the same time. Otherwise, if the interruption information is not exchanged each server time step, one model hangs up in MPI communication, because the other model was interrupted and does not answer the MPI call anymore. This implies that the server time step length needs to be a multiple of the client time step length.

- After these preparations, the contents of the MMDCLNT namelist are interpreted. The subroutine `interpret_namelist` serves two purposes:

  - The wildcards in the client *channel object* names are analysed and translated into individual *exchange fields*.
  - The namelist settings for the *mandatory fields* are cross-checked with the COSMO variables `yvarini` and `yvarbd`:
    The LOGICALs `L_INITIAL` and `L_BOUND` are set to `.TRUE.`, if the field is required by the COSMO variables. Furthermore, fields listed in `yvarini` or `yvarbd` are added to the variable `CPLDATA`. These are data fields, which are calculated by INT2COSMO, but do not require direct input from the server. For instance, the external parameters such as the root depth, the leaf area index or the orography. Note: those *mandatory fields* requiring an input field from the server side, have to be listed in the namelist. Otherwise the information about the *channel* and *channel object* name on the server side is missing.

  `CPLDATA` contains data of different sources. The first part of `CPLDATA` stems from the namelist and lists the *exchange fields*, i.e., fields that are provided by the server. During the namelist interpretation other fields are added to `CPLDATA`. These fields are calculated by INT2COSMO and are required by the COSMO model. Both types of fields are summarised by the term *coupling fields*. Therefore, two important numbers characterising `CPLDATA` are determined during this analysis:

  - `NEXCH` is the number of fields that need to be exchanged with the server.
  - `NCOPY` is the dimension of the variable `CPLDATA` containing all *coupling fields*.

  At the end of the subroutine the final list of all `CPLDATA` fields is output to the log-file.

- The MMD library includes the possibility to test the horizontal grid exchange. Therefore a *channel object* for the `test_array` (*channel* name = `'mmdclnt'`, *channel object* name = `'Test_Ar'`) is created.

- The information set in the namelist relevant for the MMD library and the server model, i.e., the *channel* and *channel object* names and the *representation* of the *exchange field*, are forwarded to the MMD library using the MMD library subroutine `MMD_Set_DataArray_Name`. The

subroutine is called for each of the *exchange fields*. Within the library, the MMD internal information structure on client and server side are set up within this subroutine and its counterpart (`MMD_Get_DataArray_Name`) on the server side. The end of the list is indicated by the presence of the optional parameter `LastEntry` which must be set to `.TRUE.` to end the list.

- **exchange_grids:** The server model automatically determines the segment of the server domain required for the interpolation in INT2COSMO using the geographical information about the client model domain. This is provided by the client within the subroutine `exchange_grids`. The local fields `rlon` and `rlat` containing the geographical coordinates of the grid points are gathered, yielding one non-decomposed field and sent to the server model. In addition, the number of *exchange fields* is sent to the server. From the geographical information the server model calculates the size of the domain, which is required to interpolate the *initial* and *boundary fields*. The grid definition for the in-coming data fields is afterwards sent back from the server to the client. The client uses this information to dimension the *in-fields* (`ptr_in`). The grid definition received from the server replaces the `grid_in` namelist of INT2LM in case of the on-line coupling. Consequently, the following (INT2COSMO) parameters are defined by the server model and sent to the client:
  - PARAMETERs describing the (rotated) server grid and the type of the soil water content and soil temperature:
    `startlat_tot`, `startlon_tot`, `endlat_tot`, `endlon_tot` , `pollat`, `pollon`, `dlat`, `dlon`, `ie_coarse` , `je_coarse`, `ke_coarse`, `ke_soil_coarse`, `itype_w_so_rel`, `itype_t_cl`. For the meaning of these variables see the server description Sect. 6.1.2[14].
  - If the server is a COSMO/MESSy model (`llm2lm = .TRUE.`), additionally the information about the vertical coordinate system and the reference atmosphere of the server COSMO model setup are required: `vcflat`, `p0sl` , `t0sl`, `dt0lp`, `delta_t`, `h_scal`, `svc1` , `svc2`, `ivctype`, `irefatm`.
  - The vertical coordinates (`vct` for ECHAM5/MESSy as server and `vcoord_in`, if COSMO/MESSy is server) and the depth of the soil layers (`czmls_in`) are exchanged.
  - Finally, the client receives two fields containing the longitude and latitude information for the *in-fields* (`latitude_in` and `longitude_in`).
- **mmdclnt_setup_int2cosmo:** This subroutine performs the setup of INT2COSMO:
  - At the beginning some switches originally determined in the INT2LM `&CONTRL` namelist are defined:
    - The LOGICALs indicating the driving model (`lgme2lm`, `lec2lm`, `lhm2lm`, `lcm2lm` and `llm2lm`) are set to `.FALSE.`; if ECHAM5/MESSy is server `lcm2lm` is `.TRUE.`; if a COSMO/MESSy model is server `llm2lm` is `.TRUE.`.
    - The LOGICAL ARRAYs `lushift_in` and `lvshift_in` indicating if and which type of a staggered grid is used for the horizontal wind components are set: For ECHAM5/MESSy as server all entries are `.FALSE.`, for the COSMO/MESSy model as server it is: `lushift_in = (.TRUE. , .FALSE.)` and `lvshift_in = (.FALSE., .TRUE.)`.
  - The INT2COSMO LOGICAL variable `linitial` is set `.TRUE.` for the very first time step of a simulation, as only for this time step initial data needs to be calculated, which is

[14]For further details about the namelist parameters see the INT2LM documentation: *http://www.cosmo-model.org/content/model/documentation/core/cosmoInt2lm.pdf*

indicated by `linitial` in INT2LM. `lcomp_bound` is another INT2LM switch, indicating that boundary data need to be calculated. It is `.TRUE.` except for the very first time step.

- Based on the "in"-grid definition and the longitudes and latitudes of the *in-fields* as received in `exchange_grids`, the staggered longitudes and latitudes (`slongitude_in` / `slatitude_in`) are calculated. As ECHAM5 does not use a staggered grid, `slatitude_in` and `slongitude_in` are set to `latitude_in` and `longitude_in`.
- Subsequently, the original INT2LM subroutine `setup_int2lm` is called. The same code is processed, apart from the initialisation and decomposition of the grid and the initialisation of many namelist parameters, which are determined directly by the setup of the COSMO/MESSy model during the on-line coupling initialisation. The changes made to the original INT2LM code in order to implement it as MESSy sub-submodel (i.e., directly coupled to COSMO/MESSy) are described in Sect. 7.
- In `setup_int2lm` the INT2COSMO namelists are read, thus the INT2COSMO LOGICAL switch `lbd_frame_cur` is set according to the namelist parameter `lbd_frame` after processing `setup_int2lm`.
- Additionally, the dimensions of the parallel decomposed *in-fields* have been calculated in `setup_int2lm`. Thus, the MMD `test_array` can be allocated by the MMD library subroutine `MMD_testC_Setup` with the corresponding dimensions.
- Finally, if ECHAM5/MESSy is the server, the hybrid coefficients for the interface levels `ak_in` and `bk_in` are set from the vertical coordinate variable `vct`, which has already been sent by the server in the subroutine `exchange_grids`. Subsequently, the hybrid coordinates for the full levels (`akh_in` and `bkh_in`) and the differences of the interface level hybrid coordinates (`dak_in` and `dbk_in`) are calculated.

At this point the initialisation of `INT2COSMO` is complete.

- **Setup_data_exchange_with_Server:** One of the crucial points of the efficient field exchange by MMD is the index list, which directly associates for each client PE the grid points of the parallel decomposed *in-field* with the grid points and PE of the parallel decomposed server grid. The index list consists of six entries for each grid point of the local client "in"-grid[15]:

  1.) the first horizontal index of the grid point in the local grid of the server PE_s (i_s)

  2.) the second horizontal index of the grid point in the local grid of PE_s (j_s),

  3.) the first horizontal index (i_c) in the local client "in"-grid,

  4.) the second horizontal index (j_c) in the local client "in"-grid,

  5.) the process entity (PE_c) on which the local client grid point is located,

  6.) the server PE (PE_s) on which the respective grid point is located[16].

  This grid association is performed by the server model. Thus the client has to send the server its grid definition and decomposition:

  - On each client PE the longitudes and latitudes of the *in-fields* `latitude_in` and `longitude_in` are written to the local fields `my_lon` and `my_lat`.

[15] i.e., the index list consists of 6 entries per number of coupled grid points: `index_list(6,number of grid points)`

[16] A more detailed example is provided in the MMD library manual, which is part of the same electronic supplement as this manual.

  - These fields are gathered on one PE in the 3-D fields (`all_lon(nx,ny,nPE)` and `all_lat(nx,ny,nPE)`) with `nx`, `ny` number of grid points in x and y direction and `nPE` number of client PEs.
  - Finally, the 3-D fields are sent to the server for further calculations.

  Due to their structure, the fields inherently contain the information required to set up the index list. The third index gives the number of the client PE and the first and second index are equal to the indices in the local grid of the respective client PE, where the point of the given geographical coordinates is located.

  After receiving this list, the server associates the (local) source points for each local client grid point to its own parallel decomposed grid and sends back the list containing the sextuples associating the client and the server grid points with each other. This list is received and analysed by the client part of the MMD library within the subroutine `MMD_C_Get_Indexlist`, which is called at the end of this subroutine (compare Fig. 3).

- **mmdclnt_set_CPLDATA:** So far, only those parts of the variable `CPLDATA` have been initialised, which are set by the namelist. In the subroutine `mmdclnt_set_CPLDATA` the POINTERs and *POINTER ARRAYs* to the data fields are set or allocated. Three different types of *coupling fields* are distinguished:

  A) fields, which require an *in-field* from the server, which is interpolated and afterwards copied to the *initial*, *boundary* or *input field*;

  B) fields exchanged with the server, which have no direct target variable: one example is the surface geopotential, which is required for interpolation itself, but has no corresponding *target field* in the COSMO/MESSy model. This is indicated by setting the corresponding client *channel* name to `'#XXX'` (see `FIELD(10)` in the `&CPL_ECHAM` namelist in Fig. 1).

  C) fields which result from the INT2COSMO interpolation/preprocessing routines but have no corresponding *in-field*. For instance, all external data fields as orography, leaf area index and so on. These fields are located at the end of the `CPLDATA` variable (indices `NEXCH+1` to `NCOPY`).

  The subroutine contains one loop over all entries of `CPLDATA`. The individual entries are indicated by the loop index `ii` in the following. The loop is split into five logical units:

  1. association / allocation of the `CPLDATA(ii)%cosmo` *POINTER ARRAY*.
  2. association / allocation of the `CPLDATA(ii)%cosmo_bd` *POINTER ARRAY*.
  3. inquiry, if the *coupling field* is an *INT2COSMO inherent field*. If 'yes',
     - the structure component `CPLDATA(ii)%lvartab` is set `.TRUE.`;
     - the `CPLDATA(ii)%vartab_name` is set and
     - the index of the variable in the INT2COSMO variable table (`CPLDATA(ii)%vartab_idx`) is set.
  4. association / allocation of the *intermediate fields* (`CPLDATA(ii)%ptr_i2c`);
  5. association / allocation of the *in-fields* (`CPLDATA(ii)%ptr_in`).

  At the beginning of the loop, the POINTERs `CPLDATA(ii)%ptr_i2c` and `CPLDATA(ii)%ptr_in` are NULLIF(Y)ied and the structure components `CPLDATA(ii)%rank`, `CPLDATA(ii)%lvartab`, `CPLDATA(ii)%vartab_name` and `CPLDATA(ii)%vartab_idx` are initialised by the default values `0`, `.FALSE.`, `''` and `0`, respectively. In the following the allocation or determination of each of the above listed `CPLDATA` entries is described in detail:

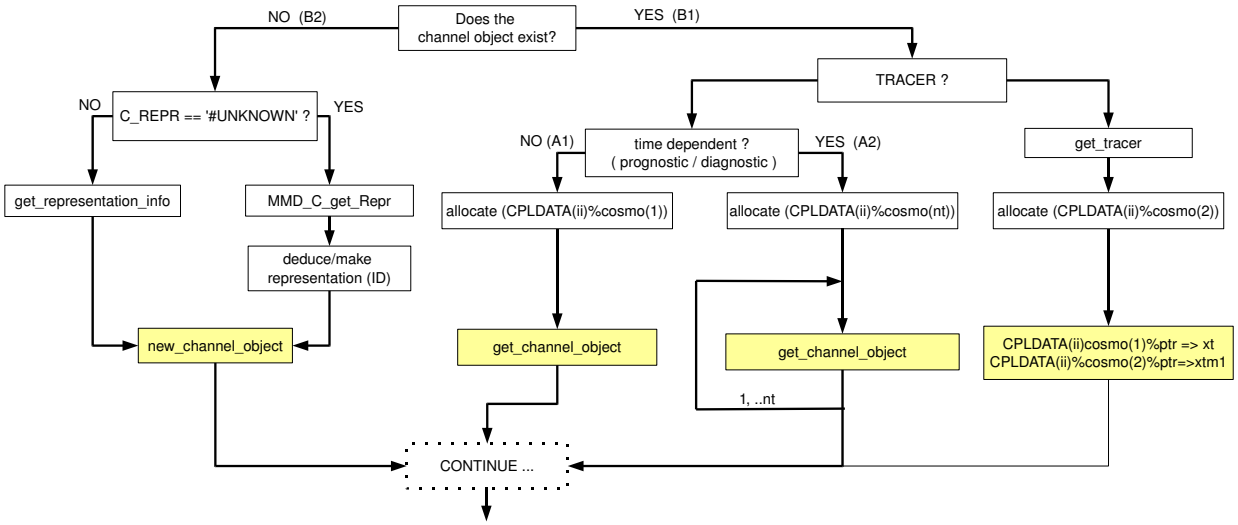

Figure 4: Flow-chart illustrating the association of the `CPLDATA(ii)%cosmo` *POINTER ARRAY*. The lables in brackets (A1, A2, B1 and B2) refer to the respective cases listed in the text. The yellow boxes point to those subroutine calls in which the individual POINTERs of the *POINTER ARRAY* `CPLDATA(ii)%cosmo` are finaly associated.

1. **Determine `CPLDATA(ii)%cosmo`:**
   This part is skipped for entries with client *channel* name `'#XXX'` as this indicates that the *exchange field* is only required in INT2COSMO.
   For the determination of the data space for the COSMO/MESSy *target field* (`CPLDATA(ii)%cosmo`) basically two times two different cases (in all combinations except B2A2) have to be taken into account:
   A) The fields can either be
      A1) diagnostic or
      A2) prognostic,
      which require different memory allocation procedures.
   B) The fields are
      B1) either already allocated by another MESSy submodel or the basemodel , or
      B2) required to be allocated within the MMDCLNT submodel itself.

   Figure 4 comprises a flow chart showing the basic procedure for the association of the `CPLDATA(ii)%cosmo` *POINTER ARRAY*

Regarding A) The nature of the respective variable (diagnostic or prognostic) determines the dimension of the *POINTER ARRAY* `CPLDATA(ii)%cosmo`:
   A1) For a diagnostic variable the dimension is 1, as only one *target field* exists.
   A2) For the prognostic variables the dimension equals the number of time levels of the time integration scheme used. For instance, for the leap frog scheme the dimension is 3, whereas for a two-time level scheme (e.g., Runge-Kutta) it is 2. This is due to two reasons:
      · For an integration scheme with more than 2 time levels, more than 1 time level needs to be initialised by MMDCLNT.

· In the COSMO model prognostic variables are allocated with an extra rank for the time level. For the sake of computational efficiency, the indices indicating the different time levels (`nnew`, `nnew` and `nold`) are rotated instead of copying the newly integrated value to the old field at the beginning of each time step. Thus, it is not a priori known which time level (index in the prognostic field) is required at a specific point in time. Hence, all time levels must be available for the coupling.

Each of the POINTERs of the *POINTER ARRAY* `CPLDATA(ii)%cosmo` (`CPLDATA(ii)%cosmo(nt)%ptr`, with `nt` being the index for a time levels) is associated to one time level of the prognostic variable. Thus, the correct time level can be addressed by the indices (`nnew` and `nnow`) usually used in the COSMO model to access the correct time levels.

Regarding B) The client *channel* name includes the information, if MMDCLNT needs to allocate the required memory itself (`'mmdclnt'` as client *channel* name in the namelist), or if the *target field* is already allocated by other COSMO/MESSy submodels or the basemodel (all other cases). In the latter case the POINTERs of the `CPLDATA(ii)%cosmo` *POINTER ARRAY* are associated to the already existing memory:

B1) The required *channel object* exists already:
First, the nature of the *channel object* is inquired by looking for the *channel object attribute* `number_of_timelevels` using the CHANNEL subroutine `get_attribute`.

A1) If the attribute does not exist, the variable is of diagnostic nature and the *POINTER ARRAY* `CPLDATA(ii)%cosmo` is allocated to the dimension 1 (as only one POINTER is required for a diagnostic variable).
Afterwards, `CPLDATA(ii)%cosmo(1)%ptr` is associated to the respective memory by calling the CHANNEL subroutine `get_channel_object`[17].
Then, the rank (`CPLDATA(ii)%rank`) of the field is acquired by calling the MMDCLNT subroutine `get_rank`. In the subroutine `get_rank`, first, the *channel object representation ID* is determined by calling the CHANNEL subroutine `get_channel_object_info` with the input parameters *channel* and *channel object* name. Second, with the *representation ID*, the `rank` of a *channel object* with this *representation* is found out by calling `get_representation_info` with the *representation ID* as input and the `CPLDATA(ii)%rank` as output parameter.

A2) If the attribute exists, the *channel object* is of prognostic nature and `CPLDATA(ii)%cosmo` is allocated to the number of time levels as denoted by the attribute (`timelev` is the return value of the subroutine `get_attribute` containing the number of required time levels). Afterwards, a loop over the number of time levels is executed associating for each time level one POINTER of the *POINTER ARRAY* to one time level of the *target field* using the subroutine `get_channel_object`. Note: for all prognostic variables individual *channel objects* for the single time levels must exist. Finally, the `CPLDATA(ii)%rank` is determined as in the diagnostic case.

XT) A special case exists for tracers:
In contrast to the prognostic COSMO variables, the tracer structure provides individual variables for all time levels[18], such that the index rotation instead of

[17] See the CHANNEL manual, which is part of the electronic supplement of Jöckel et al., 2010.
[18] Detailed information about the TRACER submodel are provided by Jöckel et al. (2008).

the copying of one time level to the other at the end of one time step is not possible for the tracers. Consequently, tracers must be treated differently:

(a) First, the tracer index `idt` is inquired by the TRACER subroutine `get_tracer`.

(b) Next, `CPLDATA(ii)%cosmo` is always allocated to 2 and the first POINTER of the *POINTER ARRAY* always points to the current tracer field `xt` of that specific tracer. The target of the second POINTER depends on the integration scheme. For a 2 time level scheme it points to `xtm1` and for a 3 level scheme to `xtf`.

(c) The rank of a tracer is always 3, thus `CPLDATA(ii)%rank` is set to 3 for tracers and

(d) the flag in the TRACER meta-structure indicating that a tracer is already initialised (`ti_gp(idt)%tp%meta%cask_i(I_MMD_INIT)`) is set to `ON` at simulation start (`lstart = .TRUE.`), otherwise the tracer field would be overwritten by subsequent tracer initialisation routines.

B2) MMDCLNT needs to allocate the memory itself, as no other submodel provides the memory for the *exchange field*. This field is calculated from an *in-field* provided by the server and supplied to other MESSy submodels (e.g. emission fields could be down-scaled from the coarse grid instead of being directly read in by IMPORT).
If the *representation* is named in the MMDCLNT namelist and stored in the structure component `CPLDATA(ii)%C_REPR`) this is an easy task. The *representation* ID `repr_input` and the corresponding rank are inquired calling the CHANNEL subroutine `get_representation_info`. Knowing the *representation* ID, the new *channel object* can be defined calling the CHANNEL subroutine `new_channel_object`. But the *representation* is not always a priori known. This is indicated in the MMDCLNT namelist by setting the *representation* string to `'#UNKNOWN'`. A classical example are emission fields provided as multi-level emissions. They are in the `Nx2D`-format (see Kerkweg et al., 2006), i.e., N levels attributed to different emission heights containing each 2-D emission information. If the number of levels is not a priori known by the client model, the server model provides additional information about the *representation*, when the *representation* string (`CPLDATA(ii)%C_REPR`) is set to `'#UNKNOWN'`. MMDCLNT acquires this information by calling the MMD library subroutine `MMD_C_Get_Repr` which provides the *representation* name (`serv_repr`), the *axis string* (`serv_axis`), the global *dimensions* (`serv_gdimlen`) and the height *attribute* (`serv_att`) of the *exchange field* in the server model. Based on this, MMDCLNT determines its own *representation*:

i) If the *representation* name is one of `'GP_3D_MID'`, `'GP_3D_INT'` or `'GP_2D_HORIZONTAL'` using the same *representation* names in the client model leads automatically to the correct result, as these are standard *representation*s.

ii) In case of the *representation* `'GP_3D_1LEV'` the *representation* is converted to `'GP_2D_HORIZONTAL'`.

iii) In all other cases MMDCLNT has to define a new *representation*:

For the definition of a new *representation* the *dimensions* need to be defined first: This is done by looping over the 4 CHARACTERS of the *axis string* `serv_axis`. Simultaneously,

- the `rank` of the array,
- the local dimension length `dim_len` and
- the *axis string* `dim_axis`

of the client array are determined. For instance, if the `il`'s component of `serv_axis` is `'X'`, the `rank` is increased by one, `dim_ids(il)` is set to `DIMID_LON`, which is the *dimension* ID for the longitude of the client model and `dim_len(il)` is set to `ie`, which is the local dimension length for the COSMO arrays.

- The horizontal dimensions of the fields are different between the server and the client model, but are implicitly given by the definition of the COSMO grid. Thus for the `'X'` and `'Y'` dimension the sizes are known and the COSMO definitions can be used.
- This is different for the `'Z'` and `'N'` dimensions, which can basically adopt every arbitrary value, but these dimensions have to be the same in the server and the client model. Thus new dimensions are defined for the `'Z'` and `'N'` dimensions, using the dimensions provided by the server model (`serv_gdimlen`). The newly defined dimensions are named in a generic way:

(a) For the vertical dimension they start with `'DIM_'` followed by a string containing the number of z-levels, and ending with `'LEV'`. For instance, when the number of z-levels is 5 this yields the name `'DIM_5LEV'`. Before actually defining the dimension, it is tested if this dimension exists already. In this case, the existing *dimension* ID is taken. This test prevents repeated definition of the same *dimension*.

(b) For the number (`'N'`) dimension the same procedure takes place, only the name of the dimension variable ends with `'N'` instead of `LEV`.

After the loop over the *axis string*, the `rank`, the local *axis string* and the local dimension lengths have been determined. This allows for the definition of the *representation*:

- If `rank` is 2 and the `'Z'` and `'N'` dimension lengths are zero, the *representation* is equal to the standard *representation* `GP_2D_HORIZONTAL`.
- If the rank is larger than 2 and smaller or equal to 4, a new *representation* needs to be defined,
- in all other cases the *representation* cannot be properly evaluated and the simulation is terminated with the error message `'CANNOT IDENTIFY REPRESENTATION'`.

The ECHAM5/MESSy model uses another order of *dimensions* as the COSMO/MESSy model. Therefore the *axis string* characters, the *dimension* IDs and the *dimension* lengths must be permutated, if ECHAM5/MESSy is server. For instance, `dim_axis = 'XZNY'` becomes `dim_axis = 'XYNZ'` and the `dim_len` has to be changed accordingly.

The new *representation*s are constructed by the subroutine `make_cosmo_representation`. Input to this subroutine are

- the lengths of the `'Z'` and `'N'` dimensions (these are zero if the corresponding *dimension* is not required),
- the *dimension* IDs (`dim_ids`),
- the *axis string* (`dim_axis`) and
- the dimension length (`dim_len`).

Output of the subroutine is the *representation* ID of the newly created *representation*. Based on the incoming parameters, the respective *representation* is created. The names of the *representation*s are as generic as the *dimension* names. If `'Z'` and `'N'` dimensions are required, the *representation* is named `'REPR_4D_zzLEV_nnN'`

where `'zz'` stands for the number of z-levels and `'nn'` for the number of n-levels. The *representation* names for `'Z'`-dimension only or `'N'`-dimension only are `'REPR_3D_zzLEV'` or `'REPR_3D_nnN'`, respectively. Using the CHANNEL subroutine `get_representation_info`, it is inquired if the *representation* exists already. In this case the return variable `reprid` is set to the ID of the matching *representation*, otherwise, the *representation* needs to be created via the channel subroutine `new_representation`. In both cases the *representation* ID is handed back to the calling subroutine. After the *representation* is identified or newly created, the new *channel object* of the `'mmdclnt'` *channel* can be created as described above for the case when the *representation* is known a priori. Additionally, a new *attribute* to the *channel object* will be set, if it was sent from the server (`serv_att`). This is required in case of Nx2D emission fields, as the layer heights have to be known in addition to the amount given by the field itself.

2. **Determine** `CPLDATA(ii)%cosmo_bd`**:**
   As for `CPLDATA(ii)%cosmo`, this part is skipped, if the client *channel* name is `'#XXX'`. If `L_BOUND` is `.TRUE.`, boundary data for the specific *coupling field* is required. In this case `CPLDATA(ii)%cosmo_bd` is allocated to 2, as the boundary layer field always consist of two layers. In the standard COSMO model, these contain the fields at the beginning and the end of the time interval for which the boundary data is valid. During this interval the boundary data is linearly interpolated according to the elapsed time. In the on-line coupled setup the two time levels for the boundary data are filled with the same values. Otherwise the server model needs to be fore-running by one boundary data time interval, which renders a 2-way nesting -as planned for the future- impossible. As the on-line coupling enables much higher coupling frequencies, the error of this procedure is small.
   Although the levels are filled with the same values and the linear interpolation in time is not required anymore, the procedure is kept in order to leave untouched as many code of the COSMO model as possible.
   For a *boundary field*, the 4D-POINTER to the full boundary data field is acquired with `get_channel_object`. Afterwards, the POINTERs of the *POINTER ARRAY* `CPLDATA(ii)%cosmo_bd(1:2)%ptr` are set dependent on the rank of the data field. For instance,

```
IF ( (CPLDATA(ii)%rank == 3  .AND.  &
 (TRIM(CPLDATA(ii)%CLIENT%CHA) /= 'tracer_gp')) THEN
   CALL get_channel_object(status              &
        , TRIM(CPLDATA(ii)%CLIENT%CHA)         &
        , TRIM(CPLDATA(ii)%CLIENT%OBJ)//'_BD' &
        , p4=bdptr                             )

...

       CPLDATA(ii)%cosmo_bd(1)%ptr =>bdptr (:,:,:,1:1)
       CPLDATA(ii)%cosmo_bd(2)%ptr =>bdptr (:,:,:,2:2)
       NULLIFY(bdptr)
```

   For a `rank=3` field the *boundary field* has 4 dimensions. Thus, the first boundary POINTER is set to the first boundary time level and the second to the second one. Note: This procedure does not work for 4-D data fields, for which *boundary fields* would be 5-dimensional.

However, a coupling to the individual boundary time levels would still be possible (and easy implementable), if required. So far such fields are not part of the model system apart from tracers.

Tracers are again processed differently. The two time levels of the boundary data are channel objects of the two TRACER CHANNELS `tracer_gp_x001` and `tracer_gp_x002`. Thus, the POINTERs to the boundary data can be associated directly by

```
CALL get_channel_object(status              &
   , TRIM(CPLDATA(ii)%CLIENT%CHA)//'_x001' &
   , TRIM(CPLDATA(ii)%CLIENT%OBJ)           &
   , p4=CPLDATA(ii)%cosmo_bd(1)%ptr         )
```

and

```
CALL get_channel_object(status              &
   , TRIM(CPLDATA(ii)%CLIENT%CHA)//'_x002' &
   , TRIM(CPLDATA(ii)%CLIENT%OBJ)           &
   , p4=CPLDATA(ii)%cosmo_bd(2)%ptr         )
```

3. **Determine CPLDATA(ii)%lvartab, CPLDATA(ii)%vartab_name and CPLDATA(ii)%vartab_idx:**
   Some data manipulations require a distinction between *INT2COSMO inherent field*s and *additional field*s. *INT2COSMO inherent fields* are all listed in the variable table structure (`var_lm`) in INT2COSMO. This table determines -among others- the *intermediate* and the *in-fields*, as well as the interpolation method. `CPLDATA(ii)%lvartab`, `CPLDATA(ii)%vartab_name` and `CPLDATA(ii)%vartab_idx` are set in a loop over the variable table of INT2COSMO:
   - The structure component `CPLDATA(ii)%lvartab` is `.TRUE.`, if the variable is element of the INT2COSMO variable table.
   - Additionally, the name of the field in the variable table is stored in the structure component `CPLDATA(ii)%vartab_name`. This is useful especially for one variable, the roughness length, as only for this the names (and the meaning) of the variables are different in INT2COSMO and in COSMO. In INT2COSMO the roughness length is called `'Z0'` whereas the COSMO model treats the product of roughness length times gravitational acceleration named `'gZ0'`. These two need to be associated with each other.
   - Finally, the location, i.e., the index, of the field in the INT2COSMO variable table is stored in the structure component `CPLDATA(ii)%vartab_idx`.

4./5. **Determine `CPLDATA(ii)%ptr_in` and `CPLDATA(ii)%ptr_i2c`:**
   The information, if a *coupling field* is part of INT2COSMO, is required for the association of the POINTERs `CPLDATA(ii)%ptr_in` and `CPLDATA(ii)%ptr_i2c`. If the field is part of the variable table, the memory for the *in-field* and the *intermediate field* have been already allocated in INT2COSMO. Otherwise the memory for these fields is allocated in MMDCLNT itself.

   a) The memory for the *intermediate* and *in-fields* exists already:
      * The POINTER `CPLDATA(ii)%ptr_in` can be directly associated calling the subroutine `get_channel_object`. For this call the object name is constructed by adding

the suffix `_IN` to the *target field* name and the name of the *channel* is `'MMDC4_IN'`[19]. When no object is found the simulation will be terminated.

- After the POINTER `CPLDATA(ii)%ptr_in` is associated, the *representation* ID of this object is obtained by calling `get_channel_object_info`.
- This ID is used to acquire the *axis string* (`CPLDATA(ii)%AXIS`) and the local dimensions (`CPLDATA(ii)%ldimlen`). This information is required by the MMD library for the data exchange.
- Afterwards, `CPLDATA(ii)%rank` is determined dependent on the third dimension of `ptr_in` to be `2` or `3`.
- Additionally, the INT2COSMO field is marked as read (`var_in(itab)%lreadin = .TRUE.`).
- The POINTER `CPLDATA(ii)%ptr_i2c` to the *intermediate field* required in INT2COSMO is set by the subroutine `get_channel_object` by using `CPLDATA(ii)%vartab_name` as *channel object* name and `'MMDC4'` as *channel* name.

b) The memory for the *intermediate* and *in-fields* still needs to be allocated (*additional fields* only):
The *representation* of the *intermediate* and *in-fields* is not a priori known. They are determined dependent on the `rank` of the field.

- For `CPLDATA(ii)%rank = 2` the *in-field* and the *intermediate field* are defined using the existing *representation IDs* `REPR_I2C_2D_IN` and `REPR_I2C_2D`, for a 2-D *in-field* and a 2-D *intermediate field*, respectively.
- If `CPLDATA(ii)%rank` is `3` and `CPLDATA(ii)%C_REPR` is `'GP_3D_MID'`, the prior defined *representation IDs* `'REPR_3D_MID_IN'` and `'REPR_I2C_3D_MID'` are used.
- In all other cases, the subroutine `make_i2c_representation` is called with the vertical and number dimensions as input parameters. The subroutine determines similarly to the subroutine `make_cosmo_representation` the *representation*s for the *intermediate field* and the *in-field*.

Using these *representation*s, the new *channel objects* for the *in-field* and the *intermediate field* are defined. The *channel objects* for the *in-fields* are added to the INT2COSMO *channel* `'MMDC4_IN'`. The *intermediate fields* are added to the *channel* `'MMDC4'` containing all *intermediate fields*.
Additionally, the *axis string* (`CPLDATA(ii)%AXIS`) and the local *dimensions* (`CPLDATA(ii)%ldimlen`) are determined by calling the subroutine `get_representation_info` with the *representation ID* `REPR_IN`.

- **Define_data_arrays:** After all data fields are associated or allocated, the respective POINTERs of the *in-fields* can be forwarded to the MMD library routines. To address the correct *exchange fields* within the MMD library, a loop over the *exchange fields* is performed by using the MMD library function `MMD_C_GetNextArray`. For each field the MMD library subroutine `MMD_C_Set_DataArray` is called, handing over the POINTER to the memory allocated for the *in-field*. Additionally, the *axis string* and the local *dimension* length are communicated to the MMD library, which, internally uses this information, later on, to unpack the data received from the server.

[19] One special case has to be considered: the *in-field* for `'W_SO'` is not necessarily named `'W_SO_IN'`, it can also be `'W_SO_REL_IN'`.

Before returning from this subroutine, the MMD library subroutine `MMD_C_SetInd_and_AllocMem` is called. It invokes the MMD internal calculation of the required buffer sizes and the actual memory allocation via `MPI_alloc_mem`.

With this the initialisation phase for the MMDCLNT submodel is complete.

5.2 Integration Phase

The procedure explained in this section is part of the subroutine `mmdclnt_init_loop`, as the update of the fields is required at the very beginning of the respective time step. The very first time step of a model simulation (`lstart = .TRUE.`) builds an exception to this rule, because in the very first initialisation not only *input* and *boundary fields*, but also the *initial fields* are calculated. The latter are already used during the end of the initialisation phase in the COSMO model. Therefore the very first data transfer and interpolation takes already place in `mmdclnt_init_memory`. Nevertheless, the procedure explained here is the same.

There are two chunks of information, which need to be exchanged during the integration phase: the coarse grid data and the TIMER status of the server.

5.2.1 Data exchange, interpolation and supply

First of all, the coupling *event* (`CPL_EVENT`) determines, if data exchange should occur in this time step. If this is the case, the MMDCLNT private subroutine `exchange_interpol_data` is called:

1. First, the data exchange is performed by calling the MMD library subroutine `MMD_C_GetBuffer`, which fills all *in-fields* with the updated values sent by the server. `MMD_C_GetBuffer` also hands back a variable containing the time in seconds, which the client had to wait until the server data was accessible. This information is written to the log-file.

2. As soon as the *in-fields* are filled, the MMD internal check of the consistency of the exchanged horizontal data field is performed. This is invoked by calling the subroutine `MMD_testC_compare`. As the content of the `test_array` does not change with time this check is only performed at the beginning (start or *restart*) of a simulation.

3. The COSMO and the INT2COSMO implementation of the MPI data exchange routines for scattering and gathering fields include dimension checks, which inhibit the exchange of data of different horizontal dimensions. But the standard 2-D- and 3-D-COSMO fields and the *in-fields* of INT2COSMO are of different horizontal resolution. This was no matter as long as COSMO and INT2LM were independent programs. In the case of on-line coupling the easiest way to cope with this, was to (re-)set the variables used for the dimension checks every time, when changing between INT2COSMO and COSMO parallelisation. This is done within the subroutine `switch_par_utilities(flag)`. When `flag=1` the check environment switches to INT2COSMO parallelisation, in the case `flag=2` it is switched back to COSMO parallelisation. Because the subroutines called in the following in `exchange_interpol_data` are INT2COSMO routines (compare flow chart Fig. 3), the parallel environment checks are switched to INT2COSMO at this place. After those more general preparations the processing of the incoming data starts:

   - **mmdclnt_prepare_external_data:** This subroutine collects all data required for the interpolation and the calculation of the *coupling fields*: INT2COSMO basically distinguishes three types of "external data":

(a) the external parameters defined on the target COSMO model grid,

(b) the external parameters as provided by the driving model (server) and

(c) the data fields provided by the driving model (server).

– External parameters are constant or slowly changing fields given as boundary conditions for the model domain, e.g., the soil type, the leaf area index, the root depth and so forth.
The external parameters for the target grid are read in from an extra file in the INT2COSMO subroutine `external_data`. Based on the read-in values, the variables required in COSMO and INT2COSMO are calculated later on. At the time being, in INT2LM all external parameters are read, meaning, even for monthly changing variables (in the climate mode of the model) the data for all twelve month is read at once. Thus reading the external parameters is only required at the beginning and the *restart* of a simulation. For the sake of higher computational efficiency, the read procedure is switched off for additional time steps in MMDCLNT_INT2COSMO. This is accomplished by the extra LOGICAL variable `lread`, which is parameter to the subroutine `external_data` and switches off the reading procedure[20].
In addition to the reading of the external parameters for the COSMO grid switched by the parameter `lread` in the subroutine `external_data`, the reading of the external parameters of the coarse grid is always omitted, because these variables are updated by the on-line data exchange. The calculation of the external parameters following the read procedure is kept virtually unchanged and the fields are processed in the same way as in INT2LM. For more details about the code changes see Sect. 7.7.

– Similarly as in `external_data`, in the subroutine `org_read_coarse_grid` the reading procedure is omitted, as the fields are already initialised by the on-line coupling.The subsequent analysis of the data, the determination of logical switches and intermediate fields is kept unchanged except for a few very small changes, which are discussed in detail in Sect. 7.16.

- **mmdclnt_interpolation:** After the preparation of the data, the interpolation begins. The interpolation of the *INT2COSMO inherent fields* proceeds exactly as in the off-line INT2LM:

  (a) The fields are interpolated horizontally by calling the INT2COSMO routine `org_coarse_interpol`. In this subroutine a field specific interpolation is performed. For 3-D-fields each vertical level is independently interpolated horizontally.

  (b) The horizontal interpolation is followed by the vertical interpolation. If the COSMO/MESSy model is the server (`llm2lm = .TRUE.`) the subroutine `org_vert_inter_lm` is called, otherwise the subroutine `org_vert_interpol`.

  (c) Subsequently, additional 2-D-fields are calculated by the INT2COSMO subroutine `org_2d_fields`.

  (d) If the server is not the COSMO/MESSy model, the fields need adjustment to the non-hydrostatic grid of the COSMO model. This is done within the subroutine `org_lm_fields`.

- **Interpol_AddiArrays:** The interpolation of the *additional fields* is based on the routines provided by INT2LM. In the subroutine a loop over all *exchange fields* (from 1 to `NEXCH`)

[20] In future it might be desirable to regularly read updated external parameters. For this we implemented the *event* `READEXT_EVENT`. Per default the adjustment of the *event* is set to `'none'`, deactivating the event, i.e., the external data are only read at start or restart of a simulation.

is processed. All fields with `CPLDATA(ii)%lvartab = .TRUE.` are skipped because they have already been interpolated in INT2COSMO. Additionally, the MMD `test_array` is excluded as the test is performed for the *in-field*.

  - In a loop over the vertical levels (or vertical levels and number dimension length for 4-D fields) the fields are interpolated horizontally by calling the subroutine `interpol_coarse_OneLayer`. This subroutine uses the weights calculated before in `org_coarse_interpol` for the *INT2COSMO inherent fields* and calls for each field the interpolation routines according to the `CPLDATA(ii)%C_INTERPOL` flags set in the namelist (as `org_coarse_interpol` does for the *INT2COSMO inherent fields*).
  - After interpolating the fields horizontally, they are vertically interpolated within the subroutine `interpol_vert_AddiArray`. Again the INT2COSMO routines are used. If the server is ECHAM5/MESSy, the interpolation routines `vert_interpol` and `vert_int_lm`, or, depending on the vertical grid `vert_z_lm` are subsequently called including the adaption to the COSMO non-hydrostatic grid. For `llm2lm = .TRUE.` (server is a COSMO/MESSy model), the subroutine `vert_interp` performs the entire interpolation. The vertical interpolation of the 4-D fields proceeds for each number dimension independently.

- After finishing the INT2COSMO routines, the dimension check is reset to the COSMO parallelisation in `switch_par_utilities`.

4. Last but not least, the *intermediate fields* calculated by the interpolation routines (`CPLDATA(ii)%ptr_i2c`) need to be assigned to the *target fields*. This task is split into two parts. The *intermediate fields* are copied to the *initial fields*, but only at the very first model time step, whereas the *boundary* and *input fields* need to be assigned each coupling time step.

   - The subroutine `move_initial_arrays_to_COSMO` moves initial data in two ways: It copies the *intermediate fields* to their *initial fields* and performs additional initialisations: In the off-line version of COSMO, the reference atmosphere and the vertical coordinate are determined within the INT2LM namelist. COSMO afterwards reads these parameters from the initial file produced by INT2LM. In the case of on-line coupling, these parameters are still determined via the INT2COSMO namelist, but the respective COSMO variables need to be set to the INT2COSMO values. Hence, the parameters determining the vertical coordinate and the reference atmosphere[21] in COSMO are set to their INT2COSMO counterparts in `move_initial_arrays_to_COSMO`.

     For the assignment of the COSMO/MESSy fields the `CPLDATA` structure is processed in one loop over the entries of `CPLDATA`. The field is skipped,

     - if `L_INITIAL` is `.FALSE.`,
     - if it is only an *exchange field* (the client *channel* name is `'#XXX'`), or
     - if it is the `test_array`.

     Otherwise, dependent on the dimension of the `CPLDATA(ii)%cosmo` the data is moved to the *target field(s)*:

     - If the dimension of `CPLDATA(ii)%cosmo` is 1, only one field needs to be assigned. Within a loop over the fourth dimension (loop index `iX`) the first three dimensions are copied:

       ```
       size3 = SIZE(CPLDATA(ii)%cosmo(1)%ptr,3)
       ```

[21] i.e., `vcflat`, `p0sl`, `t0sl`, `dt0lp`, `nfltvc`, `svc1`, `svc2`, `ivctype`, `irefatm`, `delta_t` and `h_scal`

```
CPLDATA(ii)%cosmo(1)%ptr(istartpar:iendpar   &
     ,jstartpar:jendpar,1:size3,iX) =        &
      CPLDATA(ii)%ptr_i2c(istartcos:iendcos  &
     ,jstartcos:jendcos,1:size3,iX)
```

Note: the data can only be copied on the "core-regions" (see Sect. 5.4) as only these overlap on each PE of the COSMO and of the INT2COSMO grid. This is achieved by using the indices `istartpar, iendpar, jstartpar` and `jendpar` for the COSMO grid and the indices `istartcos, iendcos, jstartcos` and `jendcos` for the INT2COSMO grid. Details about the matching of the decomposition of the two grids are provided in Sect. 5.4. As only the "core-regions" of the local domain can be initialised for each field, the COSMO subroutine `exchg_boundaries` must be called to ensure that the entire local domains are initialised.

– If the dimension of `CPLDATA(ii)%cosmo` is larger than 1, the variable depends on time and three cases are distinguished:

  * For a 2-time level integration scheme only the "new" time level needs to be initialised:

```
size3 = SIZE(CPLDATA(ii)%cosmo(1)%ptr,3)

CPLDATA(ii)%cosmo(nnew)%ptr(istartpar:iendpar &
     ,jstartpar:jendpar,1:size3,iX) =         &
      CPLDATA(ii)%ptr_i2c(istartcos:iendcos   &
     ,jstartcos:jendcos,1:size3,iX)
```

    with `nnew` being the index of the "new" time level in COSMO.

  * For a 3-time level scheme two time levels are initialised:

```
size3 = SIZE(CPLDATA(ii)%cosmo(1)%ptr,3)

CPLDATA(ii)%cosmo(nnew)%ptr(istartpar:iendpar &
     ,jstartpar:jendpar,1:size3,iX) =         &
      CPLDATA(ii)%ptr_i2c(istartcos:iendcos   &
     ,jstartcos:jendcos,1:size3,iX)

CPLDATA(ii)%cosmo(nnow)%ptr(istartpar_c4:iendpar_c4 &
     ,jstartpar_c4:jendpar_c4,1:size3,iX) =         &
      CPLDATA(ii)%ptr_i2c(istartcos:iendcos         &
     ,jstartcos:jendcos,1:size3,iX)
```

  * A special case are the tracers, as they are not accessible via index shift: The POINTERs of the *POINTER ARRAY* `CPLDATA(ii)%cosmo` are associated in a way that the first POINTER points to the `nnew` time level and the `nnow` time level is accessed by the second POINTER (see Sect. 5.1.2). Thus for a 2-time level integration scheme only the first, otherwise both fields are initialised.

  As in the diagnostic case, the copy statements are placed in a loop over the fourth dimension of `CPLDATA(ii)%cosmo(nt)%ptr` and `exchg_boundaries` needs to be called to complete the initialisation.

- The subroutine `move_boundary_arrays_to_COSMO` copies the *boundary* and the *input fields* to their respective *target fields.*

  One important difference between the boundary data association of the off-line COSMO setup and the on-line coupled setup must be emphasised here: In the off-line COSMO

boundary data are provided for two time steps (e.g., in 6 hour intervals) and the actual boundary values in-between these time steps are interpolated linearly between these two time steps. If this were to be imitated in the on-line coupling, the server model had to be one coupling time interval ahead of the client model. This could be implemented, but the ultimate goal for the development of the on-line coupling is to include a two-way nesting. In this case, the server cannot be ahead of the client model. Therefore it was decided to fill the two time levels of the boundary data with the same actual value. As the on-line setup enables a much higher coupling frequency the deviation due to the different handling of boundary data is expected to be small.

For the transfer of the *boundary* and the *input fields* a loop over the entries of `CPLDATA` is established. Again, the `test_array` and the *exchange fields* only required for the interpolation are skipped. If boundary data is requested for a field (`CPLDATA(ii)%L_BOUND = .TRUE.`) the respective *intermediate field* `CPLDATA(ii)%ptr_i2c` is copied to both time levels of the boundary data *POINTER ARRAY*:

```
size3 = SIZE(CPLDATA(ii)%cosmo_bd(1)%ptr,3)
size4 = SIZE(CPLDATA(ii)%cosmo_bd(1)%ptr,4)
CPLDATA(ii)%cosmo_bd(1)%ptr(istartpar:iendpar &
     ,jstartpar:jendpar,1:size3,1:size4) =    &
      CPLDATA(ii)%ptr_i2c(istartcos:iendcos   &
     ,jstartcos:jendcos,1:size3,1:size4)

CPLDATA(ii)%cosmo_bd(2)%ptr(istartpar:iendpar &
     ,jstartpar:jendpar,1:size3,1:size4) =    &
      CPLDATA(ii)%ptr_i2c(istartcos:iendcos   &
     ,jstartcos:jendcos,1:size3,1:size4)
```

Subsequently they are distributed to the full local domains by calling `exchg_boundaries`.

Similarly, if `CPLDATA(ii)%L_INPUT = .TRUE` for a *coupling field*, the *intermediate field* is copied to the `CPLDATA(ii)%cosmo` *POINTER ARRAY*:

```
size3 = SIZE(CPLDATA(ii)%cosmo(1)%ptr,3)
size4 = SIZE(CPLDATA(ii)%cosmo(1)%ptr,4)
CPLDATA(ii)%cosmo(1)%ptr(istartpar:iendpar   &
     ,jstartpar:jendpar,1:size3,1:size4) =   &
      CPLDATA(ii)%ptr_i2c(istartcos:iendcos  &
     ,jstartcos:jendcos,1:size3,1:size4)
```

The transfer is finalised by the boundary exchange with `exchg_boundaries`.

Before copying the *intermediate fields* to the *target fields*, one additional measure has to be taken, since some 2-D *INT2COSMO inherent fields* are flagged: The COSMO model tests for undefined values by comparing with a constant value `undefncdf` (defined as "undefined"). Usually in the COSMO off-line setup the initial and boundary data are read from files, in which undefined data points are marked by this special value. Therefore some fields are flagged during the on-line coupling for the sake of consistency:

- If the land/sea mask flag `var_lm(i)%lsm` of a variable is set to `'l'`, the variable will be flagged with `undefncdf` at points over sea.
- The grid points of `'T_SNOW'` will be set to `undefncdf` where `w_snow_lm < 0.`.

Table 1: LOGICALs switches determining the *restart* behaviour.

switch	description (if `.TRUE.`)
`lbreak`	interruption or stop of model simulation
`lstop`	stop of model simulation
`l_rerun`	write restart files
`l_TRIGGER_RESTART`	force model interruption

- `'Z0'` is multiplied with the gravitational acceleration (`g`) to get the variable `'gZ0'` as used by COSMO.

The procedure explained in detail above is summarized in Fig. 5. Furthermore it illustrates the usage of the MMDCLNT internal pointers: `ptr_in` is the *in-field*, which is input to INT2COSMO. During the horizontal interpolation the vertical and number dimensions remain untouched. The result of the interpolation is written to the *intermediate field* `ptr_i2c`. If the *in-field* is 3-D in space (number of incoming vertical levels is `ke_in`) vertical interpolation is possible. After the vertical interpolation `ptr_i2c` contains valid data on the vertical levels `1:ke` with `ke` being the number of vertical levels in the client model. After the interpolation, the *intermediate field* (`ptr_i2c`) is copied to the *target field(s)*, i.e., those variables used subsequently in the basemodel or other MESSy submodels. For *initial* and *input fields* the data is copied to the variable associated with the `cosmo(.)%ptr`, for *boundary fields* the *intermediate field* is copied to the boundary variable associated with the pointers `cosmo_bd(.)%ptr`.

5.2.2 Exchange of stop and *restart* triggers

To ensure the synchronisation of the models, information is exchanged whether the simulation is to be interrupted. Such an exchange is necessary, as, apart from the scheduled *restart*, exceptional simulation interruptions are triggered ,e.g., by QTIMER, when the available scheduler time is consumed. Table 1 lists the LOGICAL variables, which determine the behaviour of the simulation.

For the synchronisation, an exchange of this information has to take place each server time step. Thus, a `BREAK_EVENT` is set up for the client, which triggers for each server time step. If the *event* is scheduled, the subroutine `exchange_breakinfo` is called. The server sends the contents of its TIMER LOGICALs `lbreak`, `l_rerun` and `lstop`, via the MMD library routine `MMD_Inter_Bcast`. As this subroutine is not overloaded for LOGICALs, these are transferred as INTEGERs: `.TRUE.` is 1 and `.FALSE.` is 0.

`lbreak` indicates, if a simulation is going to be interrupted, `lstop` shows, if the simulation will be stopped and `l_rerun` signals, if restart file shall be written at the end of the time step. The contents of the server `lbreak` and `l_rerun` switches are directly written to the respective client switches. If the server `lstop`-switch is `.TRUE.`, this indicates that the model simulation is terminated. In this case, if the client `lstop`-switch is not yet `.TRUE.`, the client model must not directly trigger a *restart* and exit, but rather end the simulation at the `stop_date`. Thus, if the server `lServstop` is `.TRUE.` and the client itself did not yet indicate the end of the simulation (i.e., `lstop = .FALSE.`), lbreak will be reset to `.FALSE.`. This combination appears exactly in the case discussed above, when the client should continue its calculation until its own `stop_date` is reached. Additionally, if `lbreak` is `.TRUE.`, `l_rerun` and `L_TRIGGER_RESTART` are set to `.TRUE.`. Table 2 lists the contents of the LOGICALs on the client side dependent on the status of the server.

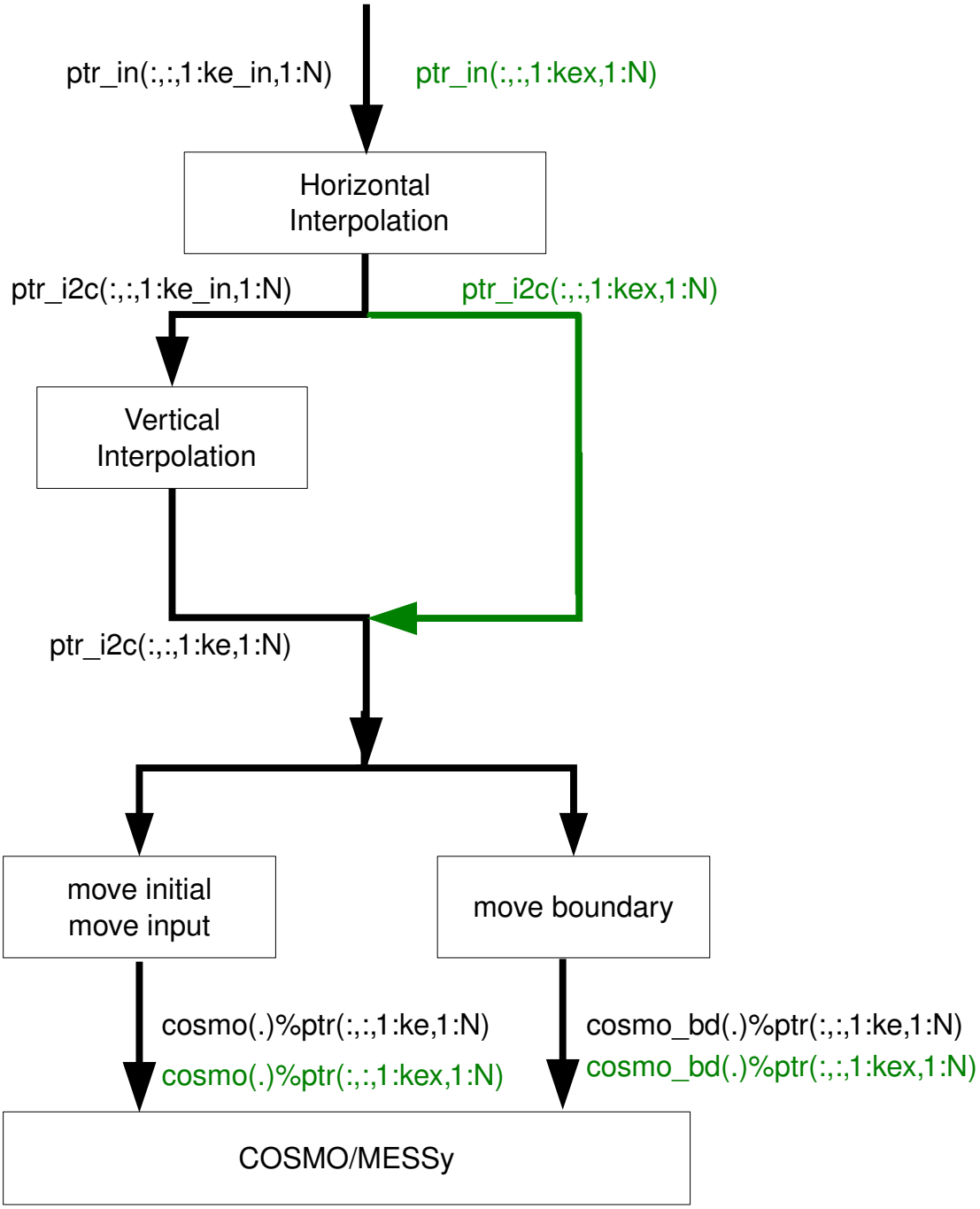

Figure 5: Pointer usage in MMDCLNT. `N` is an arbitrary (number) dimension, `ke_in` is the number of vertical levels of the *in-field*, `ke` is the number of vertical levels in the COSMO model and `kex` is an arbitrary number of vertical levels. First, the *in-field* is interpolated horizontally, second, -if requested and possible- the vertical interpolation (black) is performed and third, the *intermediate field* is copied to the COSMO/MESSy target and boundary variables.

Table 2: Contents of client LOGICALs, which determine the *restart* settings.

server status	`lbreak`	`l_rerun`
continue	`.FALSE.`	`.FALSE.`
write restart file, continue	`.FALSE.`	`.TRUE.`
restart	`.TRUE.`	`.TRUE.`
stop	`.FALSE.`	`.TRUE.`

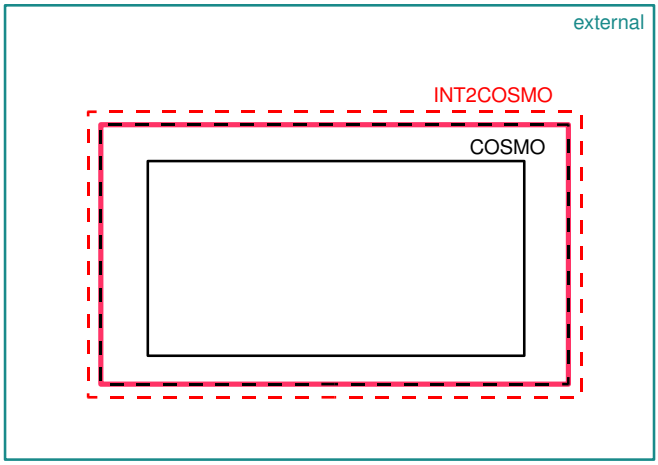

Figure 6: Illustration of the three model domains for the external parameters (turquoise), INT2COSMO (red) and the COSMO model (black). The dashed lines illustrate the entire model domains, whereas the solid lines show the "inner" domains (see text).

5.3 Finalisation Phase

After the integration phase at the end of a simulation, the memory allocated in the course of the simulation is deallocated. The MMDCLNT subroutine `mmdclnt_free_memory` releases the memory allocated by INT2COSMO by calling the INT2COSMO subroutine `org_cleanup`. For the MMD `test_array` and the other memory allocated by MMD the MMD library routines `MMD_testC_FreeMem` and `MMD_C_FreeMem` release the memory, respectively.

5.4 Grid definitions and parallel decomposition of INT2COSMO

In this section the definition of the different grids or domains used in INT2COSMO and the parallel domain decomposition of INT2COSMO, which differs from the INT2LM decomposition, are illustrated.

5.4.1 Domains

INT2COSMO works with data defined on four different domains:

1. The domain of the fields provided by the server, i.e., the "in"-grid or the *in-field* domain;

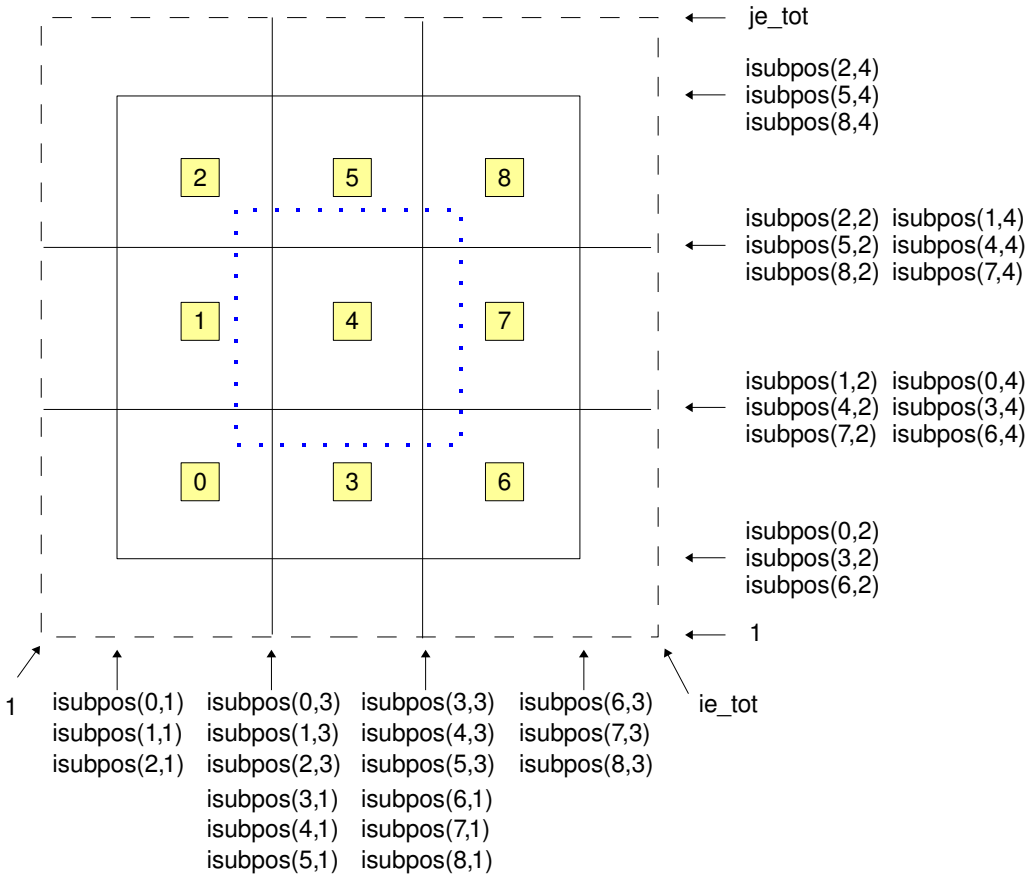

Figure 7: Example for a parallel decomposed COSMO model domain, distributed on 3x3 PEs: The numbers on yellow background are the respective PE number. The black dashed line is the border of the entire model domain, the solid lines depict the core-regions of the local domains. The blue dashed line indicates the full local domain of PE 4, i.e., including the halo or ghost boundaries. Additionally annotated are the global indices for of the core-regions.

2. the target domain, i.e., the COSMO model domain;
3. the INT2COSMO domain, i.e., the "working" domain of INT2COSMO;
4. the domain on which the external parameters such as root depth, the orography, the leaf area index or the land-sea mask, are defined.

Table 3 lists which structure components of the MMDCLNT variable CPLDATA are defined on which domain.

The *in-field* domain is defined by the server (see Sect. 6.1.2) and therefore independent of the target COSMO model domain.

The other three domains, must have the same grid spacing and need to be rotated in the same way. In other words, the only difference between these model domains is their size. The order of the domains is obvious:

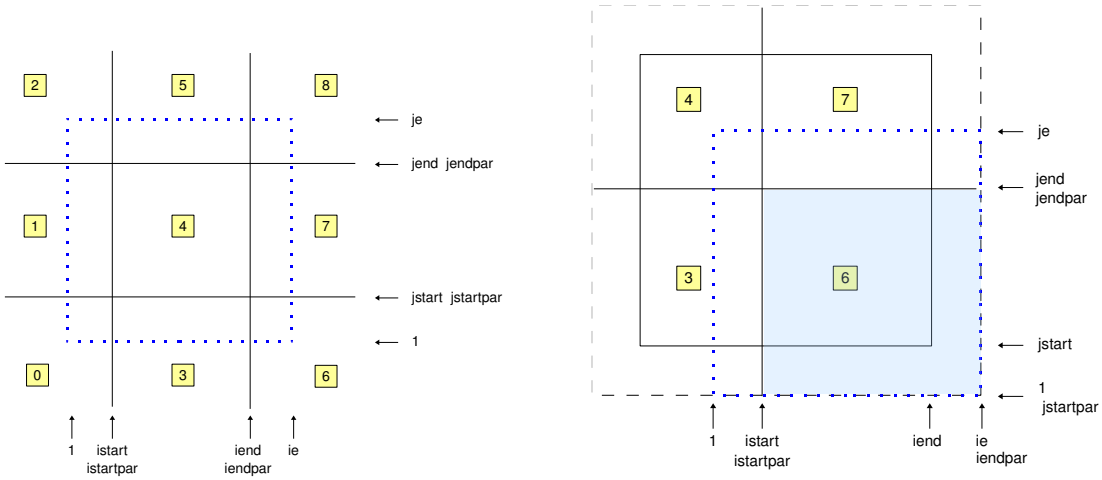

Figure 8: The figure is zooming in on PE 4 (left) and PE 6 (right) of Fig. 7, illustrating the definition of the local indices `istart`, `iend`, `jstart`, `jend`, `istartpar`, `iendpar`,`jstartpar` and `jendpar`.

Table 3: List of CPLDATA structure components and their respective model domain.

structure component	domain
`CPLDATA(ii)%ptr_in`	*in-field* domain
`CPLDATA(ii)%ptr_i2c`	INT2COSMO domain
`CPLDATA(ii)%cosmo(:)%ptr`	COSMO domain
`CPLDATA(ii)%cosmo_bd(:)%ptr`	COSMO domain

external parameter domain > INT2COSMO domain > COSMO domain

This hierarchy is caused by the processing order:

- INT2COSMO processes external parameters. Consequently, the external parameter fields need to be larger as the INT2COSMO fields.
- INT2COSMO interpolates data to the COSMO domain, thus the INT2COSMO domain has to be larger as the COSMO domain.

The model domains are illustrated in Fig. 6.

5.4.2 The parallel decomposition of the COSMO model grid and the INT2LM grid

INT2LM and the COSMO model use the same decomposition procedure. The domain is split into rectangular parts according to the numbers of PEs in x and y directions (`nprox` and `nprocy` as namelist entries). Figure 7 shows a decomposition for 9 PEs (3x3). The parts enclosed by the solid lines illustrate the so-called "core-regions" of the local (i.e. PE-bound) model domains. The sum of the core-regions covers the entire model domain, except for a frame at the lateral boundaries, which relevance becomes clear below (i.e., entire domain = "inner" domain + lateral frame).

These core-regions are unambiguously defined by their indices in the entire model domain. The indices are stored in the INTEGER ARRAY variable `isubpos(0:nPE-1,4)`, with `nPE` number of PEs. This variable lists the indices of the lower left and the upper right corner of the core-regions of each PE in the entire model domain:

- `isubpos(:,1)`: first (`'i'`) index of the lower left corner;
- `isubpos(:,2)`: second (`'j'`) index of the lower left corner;
- `isubpos(:,3)`: first (`'i'`) index of the upper right corner;
- `isubpos(:,4)`: second (`'j'`) index of the upper right corner.

Figure 7 illustrates the definition of the individual `isubpos` for the example.

To allow the computationally efficient implementation of processes, which require the data of the neighbour grid points, a "halo" or "ghost boundaries" surround the core-region of each PE. Thus, the local decomposed domain on each PE consists of the ghost boundaries and the core-region[22]. The blue dashed line in Fig. 8 illustrates the local model domains of PE 4 (left) and PE 6 (right), including the ghost boundaries and the core-region. Physical processes are calculated on the core-region, whereas the ghost boundaries are used, if the neighbouring value is required during the calculation of a physical process, e.g., for advection. Before calculating such a process, it must be ensured, that the ghost boundaries contain the correct values for the required fields. This is achieved by the subroutine `exchg_boundaries`, which transfers the data from the PE, on which a grid point belongs to the core-region, to the ghost boundary of the neighbouring PEs.

Figure 8 illustrates the definitions of the local grid and the respective indices for example PEs 4 and 6:

- The lower left corner of the local domain is always the grid point `(1,1)` and the upper right corner is `(ie,je)` with `ie` and `je` being the number of grid points in x or y direction, respectively.
- The lower left corner of the core-region is `(istart,jstart)` and the upper right corner is `(iend,jend)`.

If the PE is not completely surrounded by other PEs, the PE also hosts a part of the lateral frame of the entire model domain. In this case, the physical processes need also to be calculated on the lateral frame. The lateral frame is not a part of a core-region, but it is identical to the ghost boundaries of the respective PE. Thus, the physical processes are also calculated on the ghost boundaries which is visualised by the right hand side of Fig. 8. PE 6 is located at the lower right corner of the entire model domain. Thus, to calculate the processes on the entire model domain, on the eastern and southern border the physical processes are also calculated on the ghost boundaries. While `istart, jstart, iend` and `jend` always refer to the core-region, the indices `istartpar, jstartpar, iendpar` and `jendpar` refer to the grid points on which the physical processes are really calculated. On a PE completely surrounded by other PEs, these index quadruples are equal, as illustrated by the left hand side of Fig. 8. The right hand side of Fig. 8 shows the indices as defined for a PE at the lateral boundary. The solid black rectangle illustrates the core-region, the blue dashed line the entire local model domain. The light blue area indicates the domain part, on which the physical processes are calculated for PE 6.

[22]Note: The width of the ghost boundaries and of the lateral frame are identical. It is defined by the variable `nboundlines`

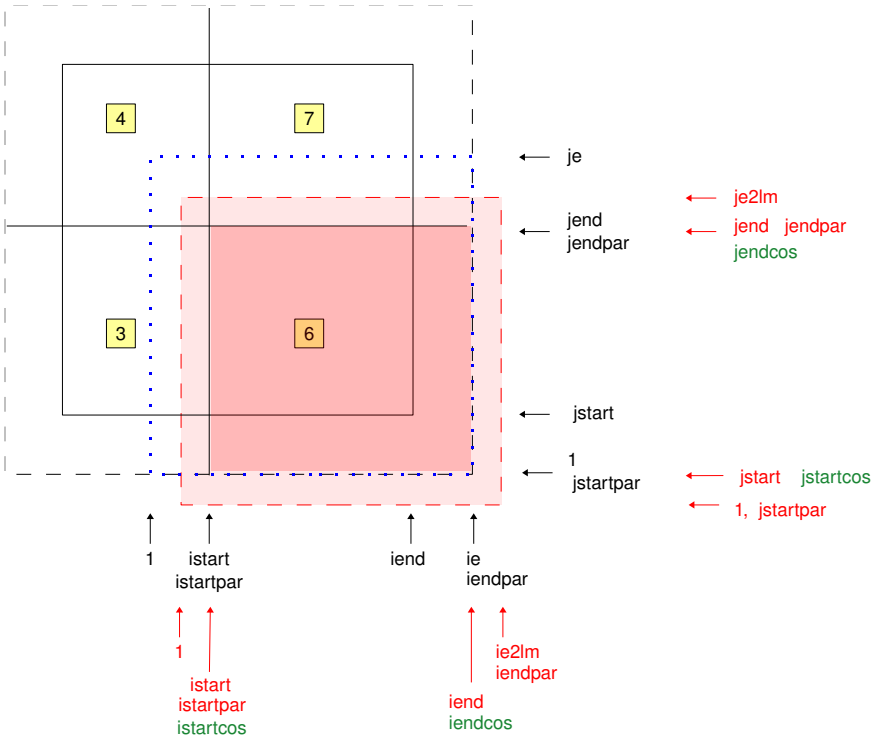

Figure 9: This is an extension of the right hand side of Fig. 8. Additionally the overlapping INT2COSMO regions for PE 6 are indicated by red rectangles. The lighter red shows the ghost latitudes, whereas the darker red indicates the INT2COSMO core-region. The red indices are the indices for the local INT2COSMO domain. Additionally, the position of the four indices `istartcos`, `iendcos`,`jstartcos` and `jendcos` are marked (green colour).
.

The width of the ghost boundaries, i.e., the number of grid points (variable `nboundlines`) added at each side to the core-region, depends on the processes taken into account in the simulation. For the COSMO model using the leap-frog time integration scheme `nboundlines`=2 is sufficient. If the Runge-Kutta time integration scheme is used, `nboundlines` is 3 or 4 depending on the order of the Runge-Kutta scheme. In contrast to this, `nboundlines` for the INT2COSMO domain is determined by the order of the interpolation routines. For the currently implemented interpolation algorithms `nboundlines` is always 1.

5.4.3 The parallel decomposition of the INT2COSMO grid

For the sake of computational efficiency, the local INT2COSMO and the local COSMO model domains should be congruent. If this is not the case, it is required to gather the fields calculated by INT2COSMO on one PE and to scatter them to the local COSMO model domain, afterwards.

Unfortunately, the decomposition algorithm used for COSMO and INT2LM does not lead to a congruent decomposition of both domains, because in the decomposition algorithm

1. an `nboundlines`-wide frame is subtracted from the entire domain,

2. the remaining domain (called "inner" domain) is almost equally distributed on the PEs, which leads to the definition of the core-regions and
3. the `nboundlines`-wide ghost-boundaries are added to the core-regions to define the entire local model domains.

The INT2COSMO domain is larger than the COSMO model domain, and the number of ghost boundaries is larger for the COSMO model than for INT2COSMO, therefore "inner" domains have always different sizes, this is why the decomposition algorithm cannot result in congruent domains for COSMO and INT2COSMO.

Nevertheless, it is possible, that the COSMO core-regions are always covered by the INT2COSMO core-regions. The coverage of the COSMO core-regions is sufficient to avoid the gather and scatter procedure for the complete fields. Nevertheless, it requires additional data exchange, for the ghost boundaries. This is performed by the COSMO subroutine `exchg_boundaries`. As the INT2COSMO core-regions are always equal are larger than the COSMO core-regions the coverage can be achieved by skilfully defining the INT2COSMO core-regions based on the COSMO core-region definition. The procedure is outlined below: The INT2COSMO domain is larger than the COSMO model domain, i.e., the entire COSMO model domain is equal to the "inner" INT2COSMO domain, i.e., the entire domain minus an `nboundlines` wide frame. (compare Fig. 6). Therefore, the INT2COSMO core-regions can be made equal to the COSMO core-regions for all PEs, except for those at the lateral boundaries. Here the INT2COSMO core-regions are larger than the COSMO core-regions. To be more precise, (following the calculation outlined below) the INT2COSMO core-regions at the lateral boundaries are equal to the parts of the local COSMO model domain described by the indices `istartpar, jstartpar, iendpar` and `jendpar`.

Coverage can only be achieved, if the core-regions of the decomposed COSMO model are taken as basis for the calculation of the decomposed INT2COSMO grid. Thus, the `isubpos` definition of the COSMO model domain (further on denoted as `isubpos_cosmo`) is taken and the INT2COSMO variable `isubpos` is calculated from this variable. For the PEs not located at the lateral boundaries of the model domain `isubpos` of the INT2COSMO domain is equal to `isubpos_cosmo+nboundlines`, with `nboundlines=1` for INT2COSMO. `nboundlines` must be added, as the global indices are shifted by `nboundlines` in the INT2COSMO grid.

In addition to this shift by `nboundlines`, at the lateral boundaries it has to be taken into account, that the "inner" INT2COSMO domain equals the entire COSMO model domain. Thus, as the `isubpos_cosmo` are the indices of the "inner" COSMO domain, these need to be shifted by `nboundlines_cosmo` in order to equal the indices of the entire model domain. Thus `isubpos` for PEs at the lateral boundaries of the INT2COSMO domain is calculated by:

$$\texttt{isubpos}(.,1) = \texttt{isubpos}_{\text{COSMO}} - \texttt{nboundlines}_{\text{COSMO}} + 1$$

at a western lateral boundary,

$$\texttt{isubpos}(.,2) = \texttt{isubpos}_{\text{COSMO}} - \texttt{nboundlines}_{\text{COSMO}} + 1$$

for a southern lateral boundary,

$$\texttt{isubpos}(.,3) = \texttt{isubpos}_{\text{COSMO}} + \texttt{nboundlines}_{\text{COSMO}} + 1$$

at a eastern lateral boundary and

$$\texttt{isubpos}(.,4) = \texttt{isubpos}_{\text{COSMO}} + \texttt{nboundlines}_{\text{COSMO}} + 1$$

at a northern lateral boundary.

Because of this procedure, the core-regions of the INT2COSMO domain and those of the COSMO model domain are not equal at the lateral boundaries. In order to copy the data from the fields defined on the INT2COSMO domains correctly to the fields defined on the COSMO model domains, additional indices are required to indicate the co-location of the COSMO local domains on which the physical processes are calculated (i.e., those indicated by `istartpar,jstartpar,iendpar` and `jendpar`) and the INT2COSMO core-regions. The indices `istartcos`, `iendcos`, `jstartcos` and `jendcos` are defined in a similar manner as `istartpar,jstartpar,iendpar` and `jendpar` by:

$$\begin{aligned} \texttt{istartcos} &= 1 + \texttt{nboundlines} \\ \texttt{iendcos} &= \texttt{ie2lm} - \texttt{nboundlines} \\ \texttt{jstartcos} &= 1 + \texttt{nboundlines} \\ \texttt{jendcos} &= \texttt{je2lm} - \texttt{nboundlines} \end{aligned}$$

with `ie2lm` and `je2lm` being the dimension of the local INT2COSMO domains (similar to `ie` and `je` for the decomposed COSMO domain). This is illustrated by Fig. 9.

6 The Server

The server fulfils three tasks, which are essential for the on-line coupling:

- It dictates the date/time and *restart* settings of the client.
- It calculates the `index_list`, i.e., the association of the client and the server grid points on the individual PEs.
- It provides the *exchange fields*.

Comparably to MMDCLNT, in MMDSERV the data for the data exchange is organised in a Fortran95 structure:

```
! CPLDATA STRUCTURE
TYPE T_COUPLE_DATA
   ! CHANNEL AND CHANNEL OBJECT NAMES IN SERVER
   CHARACTER(LEN=STRLEN_CHANNEL)         :: CHA  ! SERVER CHANNEL NAME
   CHARACTER(LEN=STRLEN_OBJECT)          :: OBJ  ! SERVER OBJECT  NAME
   ! POINTER TO SERVER  FIELD
   REAL(DP), POINTER, DIMENSION(:,:,:,:) :: ptr
   ! REPRESENTATION OF SERVER FIELD
   INTEGER                               :: rank
   INTEGER, DIMENSION(4)                 :: ldimlen=0
   ! STRING ORDER OF AXES
   CHARACTER(LEN=4)                      :: AXIS
END TYPE T_COUPLE_DATA
```

As the server only provides data, the Fortran95 structure contains considerably less components than MMDCLNT. In contrast to the client, a server can have more than one client model. Therefore the variable of TYPE `T_COUPLE_DATA` (`CPLDATA`) is itself component of a structure (TYPE `T_CLIENT_DATA`).

```
TYPE T_CLIENT_DATA
   ! NUMBER OF EXCHANGED FIELDS
   INTEGER                                          :: NEXCH
   TYPE (T_COUPLE_DATA), DIMENSION(:), ALLOCATABLE :: CPLDATA
END TYPE T_CLIENT_DATA

TYPE (T_CLIENT_DATA), DIMENSION(:), ALLOCATABLE :: CL
```

The Fortran95 variable `CL` is allocated to the number of clients of the respective server in the initialisation phase.

6.1 Initialisation Phase

6.1.1 `mmdserv_initialise`

The server inquires the number of clients it has to provide data for. The MMD library variable `MMD_Server_for_Client` is a list of the server specific client IDs in the MMD setup. For each server model it is allocated to the specific number of clients the respective server has to deal with. Thus the SIZE of this array equals the numbers of clients of the server. Subsequently, the MMD library initialisation routines `MMD_S_ALLOCATE_CLIENT` and `MMD_S_Init` are called to initialise the MMD environment of this specific server. In MMDSERV itself, the variable `CL` is dimensioned to the number of clients. As each client may require a different coupling frequency, the TIMER *event* variables `CPL_EVENT` and `CPL_IOEVENT` must also be available for each client individually.

6.1.2 `mmdserv_init_coupling`

The preparations for the coupling are all performed in the subroutine `mmdserv_init_coupling`. The subroutines described below are always processed for each client individually (indicated below by the index `ic`).

- The timing between the coupled models is synchronised in the subroutine `Setup_Client_Time`.
  1. The server receives the coupling interval in seconds from the client and defines the respective TIMER EVENT (`CPL_EVENT(ic)`).
  2. Next, the server sends its date information and its time step length to the client:
     The following dates must be exchanged to ensure that the models are synchronised: `current_date`, `start_date`, `resume_date` and `stop_date`. The dates are of TYPE `time_days`:

     ```
     TYPE, PUBLIC :: time_days
        !
        ! relative calendar date and time format
        !
        ! time_days [structure]
     ```

```
!   day    [integer]  (day in calendar, -2147483648 ..... 2147483647
!                      approx. +/-5.8 Mio. years)
!   second [integer]  (seconds of day, 0,...,86399)
!
!PRIVATE
LOGICAL :: init   = .FALSE.
INTEGER :: day    = 0
INTEGER :: second = 0
END TYPE time_days
```

Thus, each date is defined by an INTEGER indicating the day and an INTEGER for the seconds of the day. For all four dates these two INTEGERs are packed into an INTEGER array and sent to the client model. As a ninth INTEGER the time step length of the server is sent. The latter is used to define the `BREAK_EVENT` on the client side.

- After setting up the timing of the client model the MMD library subroutine `MMD_S_Get_DataArray_Name` is called, which receives the list of *exchange fields*. At this point this information is available within the MMD library, but not yet made available to MMDSERV itself. This transfer is done within the subroutine `Define_data_arrays` (see last item below).

- Before the *exchange fields* themselves are acquired, the domain section required by the client for the interpolation of the data, i.e., the domain of the *in-fields* of the client are determined in the subroutine `Setup_Client_Area`.

  - In a first step the server acquires the client grid information:
    - First, the server gets the dimensions of the client COSMO domain (`ie_tot` and `je_tot`) and the number of *exchange fields*.
    - Second, according to the dimensions, fields are allocated for the geographical longitudes and latitudes of the COSMO/MESSy grid points,
    - which are, third, sent subsequently from the client to the server.
  - Based on the client grid, the domain of the data sent to the client is determined:
    - If a COSMO/MESSy model is the server, the complete COSMO/MESSy server domain is used as *in-field* for the client. In this case simply the information about the server COSMO grid are copied to the respective transfer variables: i.e., `startlat_tot`, `startlon_tot`, `endlat_tot`, `endlon_tot`, `pollat`, `pollon`, `dlat` , `dlon`, `ie_coarse`, `je_coarse`, `ke_coarse` , `ke_soil_coarse`, `itype_w_so_rel`, `itype_t_cl`, `vcflat` , `p0sl`, `t0sl`, `dt0lp`, `delta_t`, `h_scal`, `svc1`, `svc2`, `ivctype`, `irefatm`, the vertical coordinate `vcoord` and the depth of the soil layers (`czmls`).
    - If ECHAM5/MESSy is server, a subset of the global grid is provided as input to INT2COSMO. The size of the domain is determined by the minimum and maximum longitudes and latitudes, including a check whether the date line is part of the model domain. As INT2COSMO requires a somewhat larger model domain as the COSMO grid to perform the interpolation 4.5 grid boxes are added at each side of the model domain[23]. From this model domain, the location of the corners of the grid in the parallel decomposition of the global model are calculated by the subroutine `locate_in_decomp`.

[23] The size of 4.5 is arbitrarily chosen, because it worked for the standard ECHAM5/MESSy resolutions so far employed. The smaller the difference between client and server grid the smaller this size can be chosen.

- The geographical coordinates of the South-West corner determine the start latitude (`startlat_tot`) and longitude (`startlon_tot`) and
- those of the North-Eastern corner the end latitude (`endlat_tot`) and longitude (`endlon_tot`).
- The coordinates of the rotated pole, `pollat` and `pollon`, are always `90` and `180` for a non-rotated grid like the ECHAM5 grid.
- The grid spacings in degrees for the longitudes of the global grid (`dlon`) is easily calculated by dividing `360._dp` by the SIZE of the variables containing the Gaussian longitudes of the global grid.
- The latitudes of a Gaussian grid are not equidistant, however, the grid spacings in degrees is a mandatory input to INT2LM also for the latitudes. Thus, the method implemented in INT2LM when reading and checking the netCDF-file global definitions (subroutine `read_nc_gdefs`)

  ```
  dlat = (MAXVAL(philat) - MINVAL(philat))/(ngl-1)
  ```

  is used. To minimise the error, `dlat` is recalculated after the determination of the section of the grid that is sent to the client using the maximum and minimum latitude of the sent section. Note: as the latitudes in a Gaussian grid are not equidistant this only works satisfactorily if the regional model region is not to close to the poles (with "to close" depending on the ECHAM5 resolution).
- The horizontal dimensions of the exchanged domain `ie_tot` and `je_tot` are calculated according to the difference of the corner indices.
- `ke_tot` is simply `nlev`, i.e., the number of ECHAM5 vertical levels.
- `ke_soil` is `4` as ECHAM5 includes a soil with 5 layers and 4 is the number of interfaces between the soil layers.
- The INT2COSMO variable `itype_w_so_rel`, indicating which type of soil moisture is input to INT2COSMO, must be set to `2` (for both server models).
- The variable defining the type of the climatological temperature `itype_t_cl` is set to `1`, if ECHAM5/MESSy is the server model, and to `0`, if a COSMO/MESSy model is server.
- Additionally, information about the vertical levels of the model domain i.e., the interface hybrid parameters (`vct`) for ECHAM5/MESSy and `vcoord` for the COSMO/MESSy model server are sent to the client.

∗ Last but not least, two fields containing the longitudes and latitudes for each server domain grid point are sent to the client.

- The purpose of the subroutine `Setup_data_exchange_with_Client` is to calculate the `index_list`, i.e., the list which unambiguously associates the grid points with the same geographical coordinates located in the local domains of the clients *in-field* and in the server local domain to each other. Each grid point is defined by the process number (PE) the grid point is located on, and an index pair (i, j) containing the indices of the grid point in the parallel decomposed grids. Along with determining this list the `test_array` for the MMD consistency check is filled. Thus, at the beginning of this subroutine the `test_array` is allocated by calling the MMD library routine `MMD_testS_Setup`.

  For the calculation of the `index_list` the server needs the geographic coordinates of the grid points of the local (decomposed) fields from the client. To exchange these, the server first receives three INTEGERs: the maximum dimensions for the decomposed horizontal client grid (`ie_max`

and `je_max`) and the number of client PEs. According to these dimensions, the server allocates two three dimensional fields to pick up the decomposed longitude and latitude fields sent by the client.

For each of the grid points in the horizontal domain a list member containing six entries is created. One of these sextuples consists of the client PE (PE_c), the server PE (PE_s) and the index pairs (i_c,j_c) and (i_s,j_s) of the local decomposed client and server domains, respectively, associating the two points with the same geographical coordinates in client and server grid to each other. The index information about the client grid is inherent in the longitude and latitude arrays. These fields have been gathered in a way, that the index of the third dimension corresponds to PE_c and the indices of the first two dimensions correspond to the indices in the horizontal local grid. The longitude and latitude given by the fields sent by the client are processed by the subroutine `locate_in_decomp`, which locates the respective pair of geographical coordinates on the local decomposed grid of the server model. Output of this subroutine are the PE on which the grid point is located (PE_s) and the indices in the local fields (i_s,j_s). Thus, a sextuple containing all required information about the related grid points of the client and server domain is complete. Such a sextuple is determined for each of the client *in-field* grid points yielding in a field $(i_s,j_s,i_c,j_c,PE_c,PE_s)_n$, with n number of grid points. Additionally, each sextuple is fed into the MMD `test_array` by using the MMD library function `MMD_testS_Fill`. After all grid points have been processed the filling of the MMD `test_array` is finalised by calling the MMD library function `MMD_testS_FinishFill`. The entire `index_list` containing all sextuples is forwarded to the MMD library and analysed within the MMD library routine `MMD_S_Set_Indexlist` establishing the connections between the individual server and client PEs. A more detailed explanation of this list is given in the MMD library manual, which is part of the same electronic supplement as this manual.

- Last but not least, the POINTERs to the *exchange fields* must be associated during the initialisation phase. This is achieved in the subroutine `Define_data_arrays`:
  - The structure `CL(ic)%CPLDATA`, with `ic` being the index for one client, is allocated to the number of required *exchange fields.*
  - The *channel* and *channel object* name of each field and the client *representation* as listed in the MMDCLNT namelist are retrieved by calling the MMD library function `MMD_S_GetNextArray`.
    - If the *channel* name is `'test'` the MMD `test_array` is requested and the POINTER is associated by calling the MMD library routine `MMD_testS_GetTestPtr`.
    - In all other cases the POINTER is associated by calling the CHANNEL subroutine `get_channel_object`. If the object does not exist the simulation is terminated as the required coupling is not possible.
  - When the object exists, the *representation ID* is inquired by calling the CHANNEL subroutine `get_channel_object_info`.
  - The *representation ID* must be known to subsequently retrieve the required dimension informations:
    - the *axis string*,
    - the local dimension,
    - the global dimension and
    - the *representation* name.

The latter two are required, if the *representation* name given in the MMDCLNT namelist is `'#UNKNOWN'`. In this case the client model needs the additional information plus a possible *attribute* which might contain heights (in case of multi-layer emission fields) to create the correct *representation*. This additional *attribute* is accessed by the CHANNEL subroutine `get_attribute`. All these information are forwarded to the client by the MMD library function `MMD_S_Send_Repr`.

- The POINTER, the *axis string* and the local *dimensions* are passed on to the MMD library by the subroutine `MMD_S_Set_DataArray` and processed inside to save dimension and order information for later use.
- After all data fields are processed a last call during the initialisation phase to the MMD library (subroutine `MMD_S_SetInd_and_AllocMem`) invokes the final evaluation of the dimension information in order to determine the correct buffer size. Subsequently, the actual allocation of the buffer required by MPI takes place calling `MPI_ALLOC_MEM` in the MMD library.

6.2 Integration Phase

During the integration phase, in the subroutine `mmdserv_global_start`, the server provides the *exchange fields* to its clients and informs the clients, whether the simulation is going to be interrupted.

First, the *coupling event* is tested for each client model. If the coupling with the respective client is scheduled for the current time step, the MPI Buffer is filled by calling the MMD library subroutine `MMD_S_FillBuffer`. Within this MMD library routine the data is copied to the memory buffer accessible for the client model. To ensure the correct order of accesses to the buffer from the server and the client side, the buffer is locked for that model of a client-server pair, which latest wrote to/read the buffer. If the workload of the models is not ideally balanced it happens that one of the models has to wait until it can access the buffer again. `MMD_S_FillBuffer` returns the waiting time in seconds for the server.

After the Buffer exchange, the server additionally sends the information about the status of the interrupt-switches `lbreak`, `l_rerun` and `lstop`. If the LOGICALs are `.TRUE.`, the respective entry of the INTEGER ARRAY `timeflags` is set to `1`. Otherwise it is set to zero.

6.3 Finalisation Phase

At the end of the simulation the allocated memory is released. The subroutine `mmdserv_free_memory` calls, independently for each client model, the MMD library subroutines `MMD_S_test_FreeMem` and `MMD_S_FreeMem` to release the memory allocated within the library. Additionally, the MMDSERV internal variables `CL(ic)%CPLDATA`, `CL`, `CPL_IOEVENT` and `CPL_EVENT` are deallocated.

7 Changes in INT2LM code required for the MESSy submodel INT2COSMO

All changes to the INT2LM code have been embedded in the preprocessor directive `I2CINC` (INT2COSMO IN COSMO). In this section the changes and the reasons for them are listed for each code file. The files are listed in alphabetical order.

7.1 data_fields_lm.f90 /data_fields_in.f90

As all INT2COSMO fields are allocated as MESSy *channel objects* they have to be declared as POINTERs, instead of ALLOCATABLE ARRAYs. The definitions are replaced throughout the module for all REAL variables.

Additionally, the fields `zfi_fl`, `zhi_fl`, `zps1_lm` and `zkzgr` are defined in `data_fields_lm.f90`. In the off-line INT2LM these fields are locally defined intermediate variables, which are used during the interpolation. In INT2COSMO these fields are also required for the interpolation of the *additional fields*. Therefore they are declared in `data_fields_lm.f90` and allocated as *channel objects* in `src_memory`.

7.2 data_grid_lm.f90

Instead of declaring and defining the grid in INT2COSMO independently, the vertical coordinates (`vcoord`), the number (`ke_soil_lm`) and depths (`czmls_lm` and `czhls_lm`) of the soil layers and the grid dimensions and orientation (`dlon`, `dlat`, `startlon_tot`, `startlat_tot`, `polgam`, `pollon`, `pollat`, `ielm_tot`, `jelm_tot` and `kelm_tot`) are USEd directly from the COSMO model and renamed to their INT2COSMO names:

```
USE data_modelconfig,  ONLY: vcoord, czmls_lm => czmls, czhls_lm => czhls  &
                           , ke_soil_lm => ke_soil, ielm_tot => ie_tot &
                           , jelm_tot => je_tot, kelm_tot => ke_tot    &
                           , dlon, dlat, pollat, pollon, polgam        &
                           , startlat_tot, startlon_tot
```

Furthermore, four additional index variables are defined, which are required for the grid mapping of the COSMO and the INT2COSMO grid (`istartcos`, `iendcos`, `jstartcos` and `jendcos`). The meaning of these variables is explained in Sect. 5.4.

7.3 data_int2lm_control.f90

As INT2COSMO and the COSMO model need to be set up in the same way, some run control variables are directly used from the COSMO model. Thus the declaration in `data_int2lm_control.f90` and the definition in the INT2COSMO namelist are omitted.

```
USE data_runcontrol, ONLY: nstop, nstart, llake, lradtopo, lprog_qi  &
                         , itype_calendar, idbg_level, lforest, lsso &
                         , lmulti_layer_lm => lmulti_layer
```

7.4 data_int2lm_io.f90

To make INT2COSMO as inherently consistent with the COSMO setup as possible, the variable `ydate_ini` containing the start date of the simulation is USEd from the COSMO model module `data_io` instead of being declared within this file.

7.5 data_int2lm_parallel.f90

As INT2COSMO is run in the same parallel environment as the COSMO model, most of the variables are USEd from the COSMO module `data_parallel`, instead of being defined within INT2COSMO:

```
USE data_parallel, ONLY: ldatatypes,lasync_io,nprocx,nprocy,nprocio,nproc,      &
                         num_compute, num_io,ncomm_type,my_world_id,my_cart_id, &
                         my_cart_pos,my_cart_neigh,igroup_world,icomm_world,    &
                         icomm_compute,igroup_cart,icomm_cart,icomm_row,        &
                         iexch_req,imp_reals,imp_grib,imp_integers,imp_byte,    &
                         imp_character,imp_logical,lcompute_pe,lreorder
```

7.6 data_parameters.f90

To be consistent, the KIND parameters are overwritten by those determined within the MESSy submodel `messy_main_constants_mem.f90`. `ireals` and `idouble` are set to `dp`, `iintegers` is set equal to `i4` and `isingle` to `sp`.

7.7 external_data.f90

- The MESSy submodel MMDCLNT calls the subroutine `external_data` with an additional parameter: `lread`. This LOGICAL indicates whether the external data should be read or not. When `lread` is `.FALSE.`, the initialisation of the LOGICALs (indicating the existence of specific variables in the external data file) with `.FALSE.` and the subroutine `read_lm_ext` are not processed. Additionally, `rootdp_mx` must only be set, when data was read, i.e., `lread=.TRUE.`.

- All parameters and variables determining the coarse grid are directly exchanged with the server via MMD. Thus the subroutine `read_coarse_grid_ext` is not called in INT2COSMO. The LOGICALs `lfis__in` and `lfrla_in` are set to `.TRUE.`, according to the fields exchanged during the on-line coupling.

- The vertical levels used in the calculation of the reference atmosphere are directly used from the COSMO model instead of being declared locally:

  ```
  USE data_modelconfig, ONLY : &
      klv950,       & ! k index of the LM-mainlevel, on 950 HPa
      klv850,       & ! k index of the LM-mainlevel, on 850 HPa
      klv800,       & ! k index of the LM-mainlevel, on 800 HPa
      klv700,       & ! k index of the LM-mainlevel, on 700 HPa
      klv500,       & ! k index of the LM-mainlevel, on 500 HPa
      klv400,       & ! k index of the LM-mainlevel, on 400 HPa
      klv300          ! k index of the LM-mainlevel, on 300 HPa
  ```

- The variables `fr_land_in`, `z0_in`, `plcov_in`, `plcmx_in`, `plcmn_in`, `rlaimx_in`, `rlaimn_in` and `root_in` are only deallocated from INT2LM. In INT2COSMO these variables are *channel objects* and as such automatically deallocated by the CHANNEL submodel.

7.8 src_2d_fields.f90

`t_so_lm` is defined in the vertical from `0:ke_soil+1`. As *channel objects* can only be allocated starting by 1 due to the used POINTER arithmetic, `t_so_lm` is allocated in the vertical by `1:ke_soil+2`. This has to be taken into account for the calculation of `t_so_lm`. For instance,

```
#ifndef I2CINC
          t_so_lm(i,j,0) = t_s_lm(i,j)
        ELSE
          t_so_lm(i,j,0) = undef
#else
          t_so_lm(i,j,1) = t_s_lm(i,j)
        ELSE
          t_so_lm(i,j,1) = undef
#endif
```

All places, where `t_so_lm` is calculated or used, are changed accordingly.

7.9 setup_int2lm.f90

- The memory allocation for the fields of INT2COSMO is modified (compared to INT2LM) from merely allocating the fields to defining *channel objects* for them. For the definition of *channel objects* the *representations* of these objects must be specified. Within the subroutine `make_MMDC4_channel`, which is called from `setup_int2lm`, the *dimensions* and *representations* required for the *channel objects* are created. The subroutine `make_MMDC4_channel` is located within the INT2COSMO file `src_memory.f90`.
- As INT2COSMO uses the parallelisation of the COSMO model, the subroutines `init_environment`, `init_procgrid`, `mpe_io_init` and `mpe_io_reconfig` are skipped.
- INT2LM offers the possibility to measure the timing of different phases of the interpolation process. Most of the required calls occur in subroutines and in the main program, which are skipped in INT2COSMO. Therefore, the initialisation of the timing located in `setup_int2lm` is skipped for INT2COSMO too.
- The debug output file is renamed to `'YUDEBUG_i2cinc'` because the COSMO model also writes a file named `'YUDEBUG'`.
- At the end of the subroutine the maximum length of the sent buffer (`isendbuflen`) is determined. The calculation relies on the knowledge of the decomposition of the INT2COSMO grid. In INT2LM the local domains are equally dimensioned, depending on the total number of grid boxes (`ielm_tot` or `jelm_tot`) plus the boundary lines (`nboundlines`) and the number of processes (`nprocx` or `nprocy`). This is no longer correct in INT2COSMO. Here, the local domain size also depends on the COSMO number of boundary lines (`nboundlines_cosmo`). This is taken into account in the calculation of `isendbuflen`.

7.10 src_cleanup.f90

In INT2COSMO the subroutine `free_memory` is almost completely skipped, as all fields deallocated in INT2LM in this subroutine are declared as *channel objects* and thus deallocated automatically within

the CHANNEL submodel. Only the three LOGICAL (land-sea) masks `lolp_lm`, `lolp_in` and `lmask_lm` are allocated manually in `make_MMDC4_channel` as they can not be defined directly as *channel objects* and thus are still deallocated within `free_memory`.

7.11 src_coarse_interpol.f90

- In case of INT2COSMO the on-line exchanged data is always handled similar to `'ncdf'` data. Thus, for the undefined values always the undef flag `undefncdf` is used. In INT2LM the value of the variable `undef` is determined by the type of the external data fields, i.e., `undef= undefgrib` for grib-files and `undef = undefncdf` for netCDF-files. In INT2COSMO it is possible, that the external file is in grib format, but the on-line data is processed like netCDF data. To ensure the correct setting of `undef`, it is set to `undefncdf` before the interpolation starts in INT2COSMO.
- To be able to treat the soil temperature *in-field* `t_so_in` as *channel object*, its vertical dimension is allocated by `1:ke_soil+2` instead of `0:ke_soil+1`. The change of the indexing has to be taken into account in `src_coarse_interpol.f90`, when calculating `zdt_so` and when the interpolation for the surface levels is called: i.e. `CALL interp_l(t_so_in(:,:,1),...)` instead of `CALL interp_l(t_so_in(:,:,0),...)` .

7.12 src_decomposition.f90

- As the parallel decomposition of INT2COSMO is matched with the parallel decomposition of the COSMO model, the decomposition routine for the client model (`decompose_lm`) is partly rewritten as explained in Sect. 5.4.
- The call to the subroutine `read_nc_axis` is skipped, as all information about the server model are exchanged on-line.

7.13 src_lm_fields.f90

- `zfi_fl` is an intermediate variable calculated within the subroutine `org_lm_fields`. As it is required for the interpolation of the *additional fields* as well, it is declared in `data_fields_lm` and allocated as *channel object*.
- In order to interpolate the *additional fields*, the subroutines `vert_int_lm` and `vert_z_lm` have been expanded to vertically interpolate all possible input variables and not only those specified in the code by name.

7.14 src_memory.f90

The original subroutines `alloc_lm` and `alloc_coarse_grid` are completely replaced by subroutines of the same name. Instead of allocating all fields directly, they are defined as *channel objects*. To be able to define the *channel objects*, the *dimension* IDs and the *representations* have to be created. This is done within the subroutine `make_MMDC4_channel`.

Two *channels* are created:

- The *channel* `'MMDC4'` contains the *intermediate fields* of INT2COSMO (horizontally dimensioned by `ie2lm` and `je2lm`);

- The second *channel* (`'MMDC4_IN'`) comprises the *in-fields* (horizontally dimensioned by `ie_in` and `je_in`).

In addition to the fields usually defined in `data_fields_lm` and `data_fields_in`, some fields which are only temporary fields in the INT2LM have been declared as *channel objects*, because they are required for the interpolation of the *additional fields*.

7.15 src_namelists.f90

Most of the changes in this file are due to the fact, that in INT2COSMO many INT2LM namelist switches are determined by COSMO or the server model. Thus, they must not be read in anymore. The following tables list (for each namelist) those variables excluded from the namelist. The header of the column indicates the place (basemodel or MMDCLNT) where the variables are set instead.

`&contrl`

Set by COSMO	Set by COSMO (continued)	Set by MMDCLNT	not valid for on-line coupling
ydate_ini	lprog_qi	linitial	
ydate_bd	itype_calendar	lboundaries	lante_0006
hstart	lforest	lgme2lm	lpost_0006
hstop	lsso	lec2lm	
hincbound	lradtopo	llm2lm	
nincbound	llake	lhm2lm	
nmaxwait	lasync_io	lcm2lm	
ytrans_in	lreorder	itype_w_so_rel	
ytrans_out	lmulti_layer_lm	itype_t_cl	
nprocx	ldatatypes		
nprocy	ncomm_type		
nprocio	idbg_level		

Two more `&contrl` namelist parameters are not read anymore:

- `nboundlines` is not read from the namelist as it is always 1 for INT2COSMO.
- `lmulti_layer_in` is set in accordance to `lmulti_layer_lm` for INT2COSMO.

`&grid_in`

Set by server	Set by server (continued)	Set by MMDCLNT
ie_in_tot	p0sl_in	lushift_in
je_in_tot	t0sl_in	lvshift_in
ke_in_tot	delta_t_in	
nlevskip	h_scal_in	
pollat_in	startlat_in_tot	
pollon_in	startlon_in_tot	
polgam_in	endlat_in_tot	
dlat_in	endlon_in_tot	
dlon_in	ke_soil_in	
irefatm_in	czml_soil_in	
dt0lp_in		

&lmgrid	
Set by COSMO	Set by COSMO (continued)
ielm_tot	dlon
jelm_tot	dlat
kelm_tot	startlat_tot
ke_soil_lm	startlon_tot
pollat	czml_soil_lm
pollon	czvw_so_lm
polgam	

&data
not valid/needed for on-line coupling
yinext_cat
yinext_lfn
ybitmap_cat
ybitmap_lfn
yin_cat
ylm_cat
nprocess_ini
nprocess_bd
yinext_form_read
yin_form_read

The namelists `&prictr` and `&epsctl` have not been changed. Checks required for the namelist switches are omitted, if the variables were removed from the namelist.

7.16 src_read_coarse_grid.f90

Of this module only the subroutine `org_read_coarse_grid` was modified for the implementation of INT2LM as MESSy sub-submodel INT2COSMO:

- If called from MMDCLNT, the subroutine `org_read_coarse_grid` is called without any arguments, as these are only required to read the data files, which is omitted in INT2COSMO.
- The file-type determination and read procedure dependent data blocks are skipped.
- The variable `fic_in` (control geopotential) is used for the interpolation. As it is based on *driving model* fields, it needs to be recalculated every time step at which interpolation of boundary data occurs, in order to get reproducible results. Thus, `fic_in` is calculated every time in INT2COSMO by omitting the if-statement for `var_in(mzfi_loc_in)%lreadin`.

- In INT2COSMO the *in-field* for `T_SO` is in the vertical dimension defined from `1` to `ke_soil+2` (instead of `0:ke_soil+1` in INT2LM). Thus, the surface temperature is copied to the index 1 in INT2COSMO.

```
var_in(n)%p3(1:ie_in,1:je_in,1) = &
var_in(mzts_loc_in)%p2(1:ie_in,1:je_in)
```

7.17 src_vert_inter_lm.f90

The variable `zhi_fl`, which is only temporarily calculated within the subroutine `org_vert_inter_lm` in INT2LM, is also required for the interpolation of the *additional fields*. Therefore, it is converted to a *channel object* in INT2COSMO instead of being defined locally in INT2LM. Additionally, the boundary layer height is stored in the *channel object* `zkzgr`.

7.18 src_vert_interpol.f90

In order to interpolate the *additional fields*, the intermediate variables `zps1_lm` and the boundary layer top `kzgr` are converted to *channel objects* to be available in MMDCLNT. As `kzgr` is an INTEGER and *channel objects* need to be of type REAL `kzgr` is stored in a REAL variable called `zkzgr`.

8 Changes in the COSMO code required for the on-line coupling

The COSMO model code has been changed for two reasons:

1.) The reading of the initial and boundary data files is obsolete and thus skipped, if the data is calculated on-line by the MESSy submodel MMDCLNT. The preprocessor directive `I2CINC` (INT2C OSMO IN COSMO) accomplishes this.

2.) The internal MPI environment settings need to be adjusted to the MPI environment, as required for the on-line coupling and managed by the MMD library. These changes are introduced using the preprocessor directive `MESSYMMD`.

In this section the COSMO model source files changed by these two preprocessor directives are listed and the changes are explained in detail.

8.1 Application of the preprocessor directive `I2CINC`

`src_input.f90` is the only modified file. It manages the reading of the initial and boundary data. When the COSMO model is a client, the preprocessor directive `I2CINC` prevents the opening and reading of the initial and/or boundary files:

```
#ifdef I2CINC
  ! SKIP READ-IN-PROCEDURE IN CASE OF I2CINC FOR 'initial' and 'boundary'
  IF ((ydata /= 'initial' .AND. ydata/='boundary')  &
       .OR. (.NOT. L_IS_CLIENT)) THEN
#endif
```

The variable `undef` is usually set in one of these skipped sections, for a defined preprocessor directive `I2CINC` `undef` is set at the end of the section:

```
 #ifdef I2CINC
  ENDIF
  IF (yformat /= 'ncdf') THEN
    undef     = REAL(undefgrib, ireals)
  ELSE
    undef     = REAL(undefncdf, ireals)
  ENDIF
#endif
```

In addition, specific variables are deallocated in COSMO without testing if they are really allocated. In case of `I2CINC`, the association state of the variables `iblocks`, `ibmap`, `ds_grib`, `ds_real`, `dsup`and `idims_id_in` is tested first, before they are deallocated.

8.2 Application of the preprocessor directive `MESSYMMD`

8.2.1 environment.f90

In its usual configuration the COSMO model is run in its own MPI environment. In this case the model wide communicator `icomm_world` is equal to `MPI_COMM_WORLD`. When COSMO is running within an MMD environment, it only runs on a subset of the processes of the MPI environment. Therefore, the model wide group communicator needs to be provided by MMD. This is done within the subroutine `MMD_get_model_communicator`. The subsequent use of the worldwide communicator `MPI_COMM_WORLD` would lead to errors. Thus `MPI_COMM_WORLD` was substituted by the model wide communicator `icomm_world`. To perform this substitution, the subroutine `init_procgrid` (part of the modules file `src_setup.f90`) is called with the additional parameter `icomm_world`.

The memory allocated by the MMD library needs to be released at the end of a simulation. Thus the MMD library subroutine `MMD_FreeMem_communicator` is called from the COSMO subroutine `final_environment`.

8.2.2 src_setup.f90

According to the changes in `environment.f90` the subroutine `init_procgrid` has an additional parameter (`icomm_world`), which is used instead of `MPI_COMM_WORLD` within the subroutine.

9 Changes in the ECHAM5 code required for the on-line coupling

When ECHAM5/MESSy is server in the MMD setup, the MPI environment needs to be changed accordingly. The preprocessor directive for these changes is the same as in COSMO, i.e., `MESSYMMD`.

9.1 mo_mpi.f90

- If ECHAM5/MESSy is the only executable running in an MPI environment, the communicator required to communicate with all PEs of this model is easily determined by duplicating

`MPI_COMM_WORLD` by calling the subroutine `MPI_COMM_DUP` into the model wide communicator `p_all_comm`. When ECHAM5/MESSy is running within an MMD environment, the model wide communicator `p_all_comm` is not equal to `MPI_COMM_WORLD`. Thus, the correct communicator is determined by the MMD library subroutine `MMD_get_model_communicator`.

- Before the simulation is terminated, the memory allocated by the MMD library needs to be released. This is achieved by calling `MMD_FreeMem_Communicator` from the ECHAM5 subroutine `p_stop`.

9.2 scan1.f90

One additional change had to be made, for ECHAM5/MESSy as server. The temperature is not initialised before the start of the time loop. But, when ECHAM5/MESSy is server, the first action taken in the time loop is to send the data for initialisation to the client model. Thus the temperature needs to be initialised before the first call to `messy_global_start` in case of the very first model start (`lstart = .TRUE.`). This is done by calling the ECHAM5 subroutine `initemp` before `messy_global_start` when `MESSYMMD` is defined.

Glossary

- *additional field*: An *additional field* is a field requested in the MMDCLNT namelist in addition to the fields already taken into account within INT2COSMO.
- *attributes*: *Attributes* represent time independent, scalar characteristics, e.g., the measuring unit.
- *axis string*: The *axis string* is defined for each *representation*. It indicates the order of the 'X', 'Y', 'Z' and 'N' direction, e.g., a 3-D variable in COSMO/MESSy has the *axis string* `'XYZ-'`, whereas the same variable in ECHAM5/MESSy has the *axis string* `'XZY-'`.
- *boundary field*: It is used to prescribe the variables at the model domain boundaries.
- *break event*: The *break event* is an *event* that is triggered each server time step in order to receive the information from the server, whether the server model is going to be interrupted after the current time step.
- *channel*: The generic submodel CHANNEL manages the memory and meta-data and provides a data transfer and export interface (Jöckel et al., 2010). A *channel* represents sets of "related" *channel objects* with additional meta information. The "relation" can be, for instance, the simple fact that the *channel objects* are defined by the same submodel.
- *channel object*: It represents a data field including its meta information and its underlying geometric structure (*representation*), e.g., the 3-D vorticity in spectral *representation*, the ozone mixing ratio in Eulerian *representation*, the pressure altitude of trajectories in Lagrangian *representation*.
- *coupling event*: This is an *event* scheduling the data exchange from the server to the client. Its time interval has to be a multiple of the client and the server time step length.
- *coupling field*: A *coupling field* is either an *exchange field* or a field required by the client model that is calculated during the interpolation procedure in INT2COSMO, i.e., the fields deduced from the external parameters, e.g. `lai`, `rootdp`, etc.

- *dimensions*: They represent the basic geometry of one dimension, e.g., the number of latitude points, the number of trajectories, etc.
- *event*: This is a data type provided by the generic submodel TIMER, which is used to schedule processes at specific (regular) time intervals, e.g., to trigger regular output or input during a simulation. The *event* control is part of the MESSy generic submodel TIMER. The electronic supplement of Jöckel et al. (2010) comprises a manual for TIMER and details about the *event* definition.
- *exchange field*: An *exchange field* is a field requested within the `mmdclnt.nml` namelist file and provided by the server to the client. An *exchange field* can either be a field which is interpolated and copied to a client variable, or a field required for the interpolation itself.
- *in-coming grid*: The *in-coming grid* is the grid on which the *in-fields* are defined, i.e., a subpart or the full server grid.
- *in-field*: The *in-fields* are those fields provided by the server or *driving model*, which are still defined on the server grid, but on the client side. In other words, *in-fields* are the *exchanged fields* before the interpolation.
- *driving model*: The coarse grid model (=server) that provides the *in-fields* to INT2LM / INT2COSMO.
- *INT2COSMO inherent field*: This is a field which is considered and interpolated within INT2COSMO or INT2LM (it is part of the variable table in INT2LM).
- *initial fields*: One destination type of data field provided by MMDCLNT to the client model. *Initial fields* are only used to initialise fields at the very beginning of the simulation.
- *input fields*: One destination type of data field provided by MMDCLNT to the client model. *Input fields* are *additional fields*. The newly interpolated field replaces the field in the client model, e.g., an emission field, that is down-scaled from the server.
- *intermediate field*: The *intermediate field* is the "work space" of INT2COSMO. It contains the fields after horizontal and/or vertical interpolation.
- *mandatory field*: This is an *in-field* absolutely required either by the COSMO model setup or for the interpolation itself.
- *master server*: The coarsest model in the model cascade is called the *master server*. It determines the time settings of all other model instances.
- *pointer array*: is an array of pointers of a specific dimension. For instance, a 2-D-pointer array `example_ptr` is defined by:

  ```
  TYPE (PTR_2D_ARRAY), DIMENSION(:), POINTER :: example_ptr  => NULL()
  ```

  with

  ```
  TYPE PTR_2D_ARRAY
   REAL(DP),DIMENSION(:,:),POINTER :: PTR
  END TYPE PTR_2D_ARRAY
  ```

- *representation*: It describes multidimensional geometric structures (based on *dimensions*), e.g., Eulerian (or grid point), spectral, Lagrangian.

- *representation ID*: in the CHANNEL submodel the *representation*s are stored as a list. Thus each *representation* is unambiguously identifiable by an identification number (ID).
- *restart*: A *restart* is performed, if the computing time allowed by a scheduler of a super-computer is to short to fit in the complete simulation. In this case, the simulation is interrupted in between and restarted in a new job. To achieve binary identical results for simulations with and without interruption, restart files are written, of which the contents fully determine the state of a model simulation. These files are read in the initialisation phase during a model *restart*.
- *target field*: This term specifies those fields on which the results of INT2COSMO are written, i.e., those fields used in the COSMO/MESSy simulation.

References

Jöckel, P., Kerkweg, A., Buchholz-Dietsch, J., Tost, H., Sander, R., and Pozzer, A.: Technical Note: Coupling of chemical processes with the Modular Earth Submodel System (MESSy) submodel TRACER, Atmos. Chem. Phys., 8, 1677–1687, doi:10.5194/acp-8-1677-2008, 2008.

Jöckel, P., Kerkweg, A., Pozzer, A., Sander, R., Tost, H., Riede, H., Baumgaertner, A., Gromov, S., and Kern, B.: Development cycle 2 of the Modular Earth Submodel System (MESSy2), Geosci. Model Dev., 3, 717–752, doi:10.5194/gmd-3-717-2010, 2010.

Kerkweg, A., Sander, R., Tost, H., and Jöckel, P.: Technical Note: Implementation of prescribed (OFFLEM), calculated (ONLEM), and pseudo-emissions (TNUDGE) of chemical species in the Modular Earth Submodel System (MESSy), Atmos. Chem. Phys., 6, 3603–3609, 2006.