NCREGRID - Version 1.4b
User Manual

Patrick Jöckel *(ncregrid@p-joeckel.de)

February 10, 2006

Abstract

NCREGRID is a tool for data transformation of gridded 2- and 3-dimensional (spatial) geophysical / chemical scalar fields between grids of different resolutions. The program handles data on rectangular latitude / longitude grids (not necessarily evenly spaced and / or monotone) and vertical pressure hybrid grids of arbitrary resolution. The input / output data format is netCDF. The NCREGRID core is a recursive algorithm (NREGRID) which is applicable to arbitrary (curvilinear) grids (with orthogonal axes) of any dimension and not restricted to geophysical / chemical global hybrid grids. NCREGRID is available under the GNU public license.

This manual is part of the electronic supplement of the article "Technical Note: Recursive rediscretisation of geo-scientific data in the Modular Earth Submodel System (MESSy)" in Atmos. Chem. Phys. (2006), available at: `http://www.atmos-chem-phys.org`

*Max Planck Institute for Chemistry, P.O.Box 3060, 55020 Mainz, Germany

Contents

1 Introduction

In 3-dimensional global geophysical / geochemical modelling the solving of sets of differential equations describing geophysical / chemical properties in the space time domain is often performed by discretisation with the Eulerian approach, i.e., on a discrete grid of spatial coordinates. Model input parameters need to be available on 2- or 3-dimensional (2D, 3D) grids of various resolutions. The horizontal grid space usually comprises geographical latitude and longitude. Especially in 3D global atmospheric models the vertical pressure (p) coordinate is often defined by hybrid levels (with index i) of the form

$$p(i, x, y, t) = h_a(i) \cdot p_0 + h_b(i) \cdot p_s(x, y, t) \quad , \tag{1}$$

where p_s is the surface pressure, p_0 is a constant reference pressure, and h_a and h_b are the dimensionless hybrid coefficients. This representation in a curvilinear coordinate system (dependent on longitude x, latitude y, and time t) allows a terrain following vertical coordinate, if $h_a = 0$ and $h_b = 1$ for the lowest level (surface level).

A wide variety of grid resolutions is used in different models and for different purposes. Moreover, global geophysical / chemical data (including observations) are provided in different spatial resolutions. As a consequence, data sets have often to be adapted to a specific grid resolution. For this purpose, NCREGRID was developed.

NCREGRID can be used as a stand-alone program, and / or coupled as an import interface to a model, to regrid automatically the input from an arbitrary grid space onto the required grid resolution.

The regridding is performed by a recursive algorithm, which is applicable to arbitrary orthogonal (also curvilinear) grids of any dimension, and not restricted to geo-hybrid-grid structures as described above.

2 License

3 Prerequisites

NCREGRID is written in Fortran95, thus a Fortran95 compiler is required for installation of the software. (A Fortran90 compiler which is capable to handle initialisation statements in declaration lines should do as well.) Furthermore, `(g)make` is used for building the software.

Since the input and output format is netCDF (`http://www.unidata.ucar.edu/packages/netcdf/`), the Fortran90 version (i.e., at least netCDF - Version 3.6.0) of the netCDF library is required.

NCREGRID comprises the `f2kcli` command line interface (`http://www.winteracter.com/f2kcli/index.htm`) for the stand-alone version.

4 Installation

The installation is straightforward:

1. unpack the archive:
   ```
   uncompress ncregrid.tar.Z
   tar xvf ncregrid.tar
   ```

2. change to the subdirectory `./ncregrid`:
   ```
   cd ncregrid
   ```

3. configure NCREGRID according to your system:
   `./configure` *VARIABLE=VALUE ...*
   with possible *VARIABLEs*:

`F90`	name of the Fortran90/95 compiler	(optional)
`F90FLAGS`	Fortran90/95 compiler options (e.g., option for invoking the cpp preprocessor)	(required)
`NC_INC`	(absolute) path where `netcdf.mod` is located	(required)
`NC_LIB`	(absolute) path where `libnetcdf.a` is located	(required)

   Platform / Compiler specific notes can be found in the `README` file of the distribution.

4. build the executable and the modules:
   ```
   gmake
   ```

5. move the executable and the modules to the `./bin` and `./include` subdirectories, respectively:
   ```
   gmake install
   ```

6. clean-up the source directory:
   ```
   gmake clean
   ```

The executable `ncregrid` should now be available in the `./bin` subdirectory, the modules in the `./include` subdirectory. The status after unpacking the archive can be achieved with `gmake distclean`.

5 Tested Systems

NCREGRID has been tested so far on / with the following platforms / compilers:

- PC (i686) Linux-2.4.18 (SuSE 8.0 - 10.0) with
  lf95 (Lahey/Fujitsu Fortran 95 Express Release L6.10a)
- Dec-Alpha OSF1-4.0f with
  f90 (Compaq Fortran 90 (formerly DIGITAL Fortran 90))
- Dec-Alpha OSF1-5.1a with
  f90 (Compaq Fortran 90 (formerly DIGITAL Fortran 90))
- Silicon Graphics Origin 2100
- Windows 98 with Compaq Visual Fortran
- IBM-AIX
- SGI Origin 3800, IRIX6.5 with MIPSpro 7
- NEC-SX6
- PC (i686) Linux-2.4.20 (SuSE 8.2) with
  NAG-f90
- Apple G5, OS v10.4.4 with xlf90 v8.1

If you succeed to build NCREGRID on platforms which are not listed here, please send an e-mail with system / compiler information, potential changes / extensions to the source code, and / or the Makefile to `ncregrid@p-joeckel.de`.

6 Usage

NCREGRID can be applied in two different modes:

- Stand-alone mode for the rediscretisation of data files in netCDF
- Interface mode (coupled to a model) for automatic rediscretisation during data import from netCDF files

In both modes, NCREGRID is basically applied in 4 steps:

1. Analysis of the netCDF file containing the data which should be regridded (*infile*). This can for instance be done with:
   `ncdump -h` *infile*
   Required are the names of the netCDF variables spanning the grid (such as latitude, longitude, surface pressure, reference pressure, hybrid-coefficients, time), and the names of the variables which should be regridded. Note that this step is left to the user, in order to be independent of specific netCDF conventions. Note further that netCDF and NCREGRID are case-sensitive.

2. Analysis of the output grid structure:

   - The output grid structure is available in a netCDF file:
     The information from that file (*grdfile*) can be extracted from this file in analogy to step 1. above.
   - NCREGRID is used in interface mode:
     The output grid information is provided by the grid-specification interface routine (see section 9.1)
   - The output grid structure is not available:
     The output grid information can be written with an appropriate text editor in the CDL-syntax (see netCDF-manual). From this, a netCDF file can be generated by
     `ncgen -b -o` *grdfile cdl-file*
     and used as *grdfile*, as above.

3. Summary of all required information in a namelist:
   The NCREGRID-namelist structure is described in detail in section 8.

4. Start NCREGRID:

   - NCREGRID in stand-alone mode:
     `ncregrid` *namelist-file*
   - NCREGRID in interface mode:
     start the executable with NCREGRID linked
     Detailed information about the usage of NCREGRID from Fortran90 / Fortran95 code is provided in section 9.

7 Regridding Types

The core of NCREGRID, NREGRID (see section 10.4), is designed to rediscretise arbitrary distributions from one grid to another, without adding information (e.g., if regridding from a coarser to a finer grid), or reducing more information than is lost anyway (e.g., if regridding from a finer to a coarser grid). **Thus, no inter- / extrapolation methods are applied!** The data fields are purely redistributed,

assuming constant values within the grid box, represented by the grid box mid point. For this, intensive and extensive variables are distinguished.

Accordingly, the following regridding types (RG_TYPE) are supported at present for the regridding of variables (scalar fields!) between geo-hybrid grids:

- INT is suitable for intensive quantities, such as tracer mass mixing ratios, or temperature fields. This option conserves the global weighted average of a scalar field during the regridding procedure.
- EXT is suitable for extensive quantities, such as tracer masses, or tracer emission maps (2D). This option conserves the global unweighted sum over all grid boxes of a scalar field.
- IDX is suitable for regridding index distributions, i.e., scalar fields with discrete values, where averaging is not defined. The index-regridding returns for a given destination grid-box the index, which has in all overlapping source grid-boxes the largest relative contribution.
- IFX is similar to IDX, however it returns a variable extended by one dimension, whereby the additional dimension is along the index range. A given slice along the new index-dimension contains the fraction of that index in the respective grid-box.

8 Namelist Control

The regridding procedure of NCREGRID is controlled by a namelist (and some optional control parameters specified at the subroutine call of `REGRID_CONTROL`, if NCREGRID is used in interface mode). The syntax of a NCREGRID-namelist is:

```
&REGRID
variable = value,
...
/
```

The dots indicate a list of further namelist entries. Several namelists can be concatenated into one *namelist-file*. These namelists are then processed by NCREGRID in sequence. Table 1 gives an overview of the possible namelist entries with their meanings.

8.1 Grid specification

In order to allow netCDF-files to be as general as possible, and not to be restricted to specific netCDF-conventions, the input grid (read from the netCDF file *infile*) has to be specified by the user via the namelist. With the namelist variables `i_lat(i/m)`, `i_lon(i/m)`, `i_hya(i/m)`, `i_hyb(i/m)`, `i_ps`, and `i_p0` a 3D input grid (spatial) is fully described. The output grid (`g_...`) is specified in analogy, and read from the *grdfile*. The output is written to the netCDF file *outfile*.

INPUT	OUTPUT	Description
infile	outfile	input/output netCDF file
grdfile		netCDF file with output grid
i_lat(m/i)	g_lat(m/i)	name of latitude axis
i_latr	g_latr	range of latitude axis
i_lon(m/i)	g_lon(m/i)	name of longitude axis
i_lonr	g_lonr	range of longitude axis
i_hya(m/i)	g_hya(m/i)	name of hybrid-A-coeff. (h_a)
i_hyar	g_hyar	range of hybrid-A-coeff.
i_hyb(m/i)	g_hyb(m/i)	name of hybrid-B-coeff. (h_b)
i_hybr	g_hybr	range of hybrid-B-coeff.
i_time(m/i)	g_time(m/i)	name of time axis
i_ps	g_ps	name/value of surface pressure
i_p0	g_p0	name/value of reference pressure
i_t	o_t	input/output time step control
g_t		grid time step control
var		variable list
pressure		vert. regridding in pressure coord.
input_time		output file gets input time axis

Table 1: List of NCREGRID namelist variables. With (...m/...i) the box mid point coordinates (...m) and/or the box interfaces (...i) are specified, respectively. i_t(4), o_t(3), and g_t(3) are integer vectors, pressure and input_time are of type logical, i/g_latr, i/g_lonr, i/g_hyar, and i/g_hybr are real vectors of length 2, all others are character/string variables.

For the regridding procedure, the interfaces (i...) of the grid boxes are required (except for i/g_timei). If the respective data is not available in the *infile* / *grdfile*, or the entries are not present in the namelist, the interface values are internally calculated from the corresponding mid-box values, assuming that the interfaces are half-way between the mid-points. The outer interfaces are calculated using the same distance between outermost mid-point and corresponding inner interface. If this calculation of the outer interfaces is not applicable, the user can specify them with the namelist variables i_latr, i_lonr, i_hyar, i_hybr for the input grid, and with g_latr, g_lonr, g_hyar, g_hybr for the output grid, respectively. For example,

```
...
i_latm = 'latitude',
i_latr = -90.0, 90.0,
...
```

in the namelist ensures that the outer input grid latitude interfaces (calculated from the mid-box latitudes with name latitude) are at $-90.0°$ and $90.0°$. Note that in case of i/g_hyar the order of parameters is relevant, since the hybrid-A-coefficients (see Eq. (1)) are not monotone. Therefore, in

```
i_hyar = a1, a2,
```

a1 refers to the highest grid level (corresponding to the smallest hybrid-B-coefficient (!)), and a2 to the lowest grid level (corresponding to the largest hybrid-B-coefficient (!)), respectively.

Since the core of NCREGRID (NREGRID, see section 10.4) performs regridding in rectangular grid space without weighting, the latitude axis intervals have to be weighted according to the surface. Therefore, NCREGRID uses internally

$$\sin(\text{latitude}) \tag{2}$$

for regridding along the latitude axes (see `messy_ncregrid_geohyb.f90`, `SUBROUTINE GEOHYBGRID_AXES`).

If the interface variables are specified in the namelist, but not the corresponding mid-points, the latter are internally calculated, assuming that the mid-points are half-way between the interfaces. The mid-points, however, are not used by the regridding procedure.

Dimensions defined for the input grid (*infile*), but omitted for the output grid (*grdfile*) are treated as invariant (see section 10.2).

8.2 Syntax of the namelist variable var

With the namelist variable `var` the user specifies the scalar fields of the *infile* (which have to be on the specified input-grid) that should be regridded. For output to the *outfile*, the variables can be renamed and scaled. Moreover, the regridding type (see section 7) can be assigned. The syntax is

```
var = '[new_name=]name[:RG_TYPE][,scale]; ...',
```

whereby the dots indicate a list of further variables. *name* is the variable name in *infile*, *new_name* is the variable name in *outfile*, *RG_TYPE* is the regridding type (see section 7), and *scale* is the scaling factor. The order of *:RG_TYPE* and *,scale* is arbitrary.

If *new_name* is omitted, the variable is not renamed. If *scale* is omitted, the variable is not scaled. If *RG_TYPE* is omitted, NCREGRID checks the *infile* for the variable attribute `RG_TYPE`. If this attribute is not set, or the value is not recognised, NCREGRID takes `INT` as default (see section 7).

If the namelist variable `var` is not specified at all, NCREGRID scans the *infile* for all variables on the specified input grid. Renaming and scaling are not performed. The regridding type is set to `INT`, unless the variable attribute `RG_TYPE` in *infile* is specified.

8.3 Time control

With the time control namelist variables `i_t`, `g_t` and `o_t` the user specifies the time-steps of netCDF-variables for regridding. The syntax is

```
i_t = itmin,itstep,itmax,itret,  ! default: 1, 1, 0, 0
g_t = gtmin,gtstep,gtreset,      ! default: 1, 1, 0
o_t = otstart,otstep,otdummy     ! default: 1, 1, 0
```

where *itmin*, *itstep*, *itmax*, *itret*, *gtmin*, *gtstep*, *gtreset*, *otstart*, and *otstep* are integers The default settings are listed above. The third entry of `o_t` (*otdummy*) is currently not used.

NCREGRID resets automatically *itstep*, *gtstep*, and / or *otstep* to 1, if 0 is specified in the namelist; *itmax* and *itret* are set to *itmin*, if not specified in the namelist. The overall consistency of all time step control parameters is checked by NCREGRID, and documented by the output of error / warning messages, if required.

The input variables are regridded between *itmin* and *itmax* with a step size of *itstep*. The regridded data of time step *itret* are returned to the `REGRID_CONTROL` subroutine call (see section 9.2). Thus, *itret* has no meaning in the stand-alone mode of NCREGRID.

For the destination grid of the variables at the input time steps (*itmin*, *itmin+itstep*, *itmin+2×itstep*, ..., *itmax*) the grid from *grdfile* is used at time step *gtmin*, *gtmin+gtstep*, *gtmin+2×gtstep*, ... , respectively. If *gtreset* is reached, the *grdfile* time step is reset to *gtstart* again, etc. (This allows for instance the regridding of 60 time steps of monthly averaged data (in *infile*), to a grid (in *grdfile*) which is only known climatologically, i.e., containing 12 monthly averages.)

And finally, the regridded data is written to the output file with time steps *otstart*, *otstart+otstep*, *otstart+2×otstep*, ... This is needed, e.g., if *itstep* is not 1, but the *outfile* should contain a continuous time series.

With the namelist variable `input_time` the time axis specification in the output file (*outfile*) is set. Per default (`input_time = T`) the *outfile* time axis is the same as in *infile*. Otherwise, (`input_time = F`) the *grdfile* time axis (or interface time axis in interface mode) is taken for *outfile*. Additionally, in interface mode the output time stepping (`o_t`) is set to the *infile* time stepping (in case of `input_time=T`), or to the interface time stepping (in case of `input_time = F`), respectively.

8.4 Vertical axis specifications

NCREGRID is capable to handle all cases of vertical pressure axes, such as

- hybrid pressure axes ($h_a \neq 0$, $h_b \neq 0$);
- constant pressure axes ($h_b = 0$); `i/g_hybi/m` omitted in namelist
- sigma levels ($h_a = 0$); `i/g_hyai/m` omitted in namelist

8.4.1 Surface and reference pressure

If the surface pressure and/or reference pressure is / are not contained in *infile* and / or *grdfile*, respectively, or they should not be used, it is possible to specify a constant value, for example:

```
i_p0 = '101325.0 Pa'
```

The syntax is the same for `g_p0`, `i_ps`, and `g_ps`. Note that in these cases the unit must be chosen such that surface pressure, reference pressure and the hybrid-coefficients are consistent (see Eq. (1)). The unit ('Pa' in the specification above) is only converted to the netCDF variable attribute `units` in the output file, but not used internally for automatic unit conversions!

8.4.2 Vertical regridding coordinates

Calculation of the vertical overlap of grid-boxes between *infile* and *grdfile* is internally performed in sigma-coordinates per default

$$\sigma(i) = p(i, x, y, t)/p_s(x, y, t) \ , \tag{3}$$

in order to avoid conservation problems in case the source and destination surface pressure fields are different. However, a vertical regridding in pressure coordinates can be enforced, if the variable `pressure = T` is specified in the namelist. Note, however, that in such cases input and output pressure levels must have the same units!

8.5 Namelist examples

Examples of NCREGRID namelists can be found in the `./ncregrid/examples/namelist` subdirectory of this distribution.

9 Interface Mode

The usage of NCREGRID in interface mode, i.e., linked to another program, requires two steps:

- Specification of the destination grid in the Fortran95 code, and
- calling the regridding procedure from the Fortran95 code. This can be achieved by using the level 1 and level 2 interface routines described below.

9.1 Destination grid specification

For NCREGRID in interface mode (e.g., as part of a model) a specification of the destination grid in the Fortran95 code is required, which provides the information as alternative to the specification via the `grdfile` entry in the namelist (see section 8). This definition section is contained in the module `messy_ncregrid_interface.f90`. An example can be found in the `./ncregrid/examples/interface` subdirectory. The grid definition module provides two subroutines:

- `SUBROUTINE INTERFACE_GEOHYBGRID`: returns the grid information as a Fortran95 structure (`TYPE geohybgrid`, see `messy_ncregrid_geohyb.f90`, section 10.3)

- SUBROUTINE INTERFACE_SET_MESSAGEMODE: controls the degree of diagnostic output of NCREGRID; The following parameters can be used, i.e., summed (MSGMODE = ... + ... + ...):

MSGMODE_S	silent
MSGMODE_E	error messages
MSGMODE_VL	little verbose
MSGMODE_W	warning messages
MSGMODE_VM	medium verbose
MSGMODE_I	info messages

9.2 Level 1 interface for calling NCREGRID

The level 1 interface routine for calling NCREGRID from Fortran95 code provides access to all controllable features: SUBROUTINE REGRID_CONTROL. The following example shows a typical application:

```
!------------------------------------------------------------------------
SUBROUTINE MY_SUBROUTINE(...)

  ! NCREGRID INTERFACE
  USE messy_ncregrid_control, ONLY: RG_CTRL, RG_NML, RG_STATUS  &
                                  , RG_SCAN, RG_PROC, RG_STOP   &
                                  , NML_NEXT,NML_STAY           &
                                  , RGSTAT_START, RGSTAT_CONT   &
                                  , RGSTAT_STOP                 &
                                  , REGRID_CONTROL
  USE messy_ncregrid_nc,      ONLY: ncvar
  USE messy_ncregrid_geohyb,  ONLY: geohybgrid

  ! (OPTIONAL) OUTPUT OF REGRIDDING PROCEDURE
  TYPE (ncvar), DIMENSION(:), POINTER      :: var   ! list of variables
  TYPE (geohybgrid)                        :: grid  ! output grid info

  ! MY VARIABLES
  CHARACTER(LEN=100) :: filename  ! name of namelist file
  ! ...

  ! MY CODE

  ! NAME OF FILE WITH NAMELIST(S)
  filename = 'test.nml'

  ! START REGRIDDING
  ! EXAMPLE: REGRIDDING ALL VARIABLES IN ALL NAMELISTS
  RG_CTRL = RG_PROC   ! IMMEDIATE REGRIDDING
  RG_NML  = NML_NEXT  ! READ NEXT NAMELIST FROM FILE
  !
  DO ! endless DO loop (must be terminated with EXIT)
```

```
     CALL REGRID_CONTROL(RG_CTRL, RG_NML, RG_STATUS    &
                         ,TRIM(filename)               &
                        !,iounit = 10                  & ! DEFAULT = 17
                        !,lrgx=..., lrgy=..., lrgz=... & ! DEFAULT:
                        !,tmin=..., tmax=...           & ! namelist
                        !,tstep=..., tret=...          & ! entry
                         ,var = var                    &
                         ,grid = grid                  &
                         )

    IF (RG_STATUS == RGSTAT_STOP) THEN
        WRITE(*,*) 'END OF NAMELIST FILE REACHED !'
        EXIT
    END IF

    ! USE THE REGRIDDED VARIABLES (var) AND THE OUTPUT GRID (grid) HERE

  END DO
  ! END REGRIDDING

END SUBROUTINE MY_SUBROUTINE
!------------------------------------------------------------------------
```

In the second example, regridding is performed only on special conditions, whereby these conditions depend on the variables or the grid of the *infile*. This can be used, e.g., if only a special subset of variables should be regridded, whereby the subset depends on the model state.

```
!------------------------------------------------------------------------
SUBROUTINE MY_SUBROUTINE(...)

  ! NCREGRID INTERFACE
  USE messy_ncregrid_control, ONLY: RG_CTRL, RG_NML, RG_STATUS  &
                                  , RG_SCAN, RG_PROC, RG_STOP   &
                                  , NML_NEXT,NML_STAY           &
                                  , RGSTAT_START, RGSTAT_CONT   &
                                  , RGSTAT_STOP                 &
                                  , REGRID_CONTROL
  USE messy_ncregrid_nc,      ONLY: ncvar
  USE messy_ncregrid_geohyb,  ONLY: geohybgrid

  ! (OPTIONAL) OUTPUT OF REGRIDDING PROCEDURE
  TYPE (ncvar), DIMENSION(:), POINTER      :: var   ! list of variables
  TYPE (geohybgrid)                        :: grid  ! output grid info

  ! MY VARIABLES
  CHARACTER(LEN=100) :: filename  ! name of namelist file
  ! ...
```

```
! MY CODE

! NAME OF FILE WITH NAMELIST(S)
filename = 'test.nml'

! START REGRIDDING
! EXAMPLE: REGRIDDING CONTROLLED BY INPUT VARIABELS OR
!          INPUT-GRID (BOTH SPECIFIED IN NAMELIST)
RG_CTRL = RG_SCAN   ! SCAN-MODE, NO REGRIDDING
RG_NML  = NML_NEXT  ! READ NEXT NAMELIST FROM FILE
!
DO ! endless DO loop (must be terminated with EXIT)

   CALL REGRID_CONTROL(RG_CTRL, RG_NML, RG_STATUS    &
                       ,TRIM(filename)               &
                      !,iounit = 10                   & ! DEFAULT = 17
                      !,lrgx=..., lrgy=..., lrgz=... & ! DEFAULT:
                      !,tmin=..., tmax=...           & ! namelist
                      !,tstep=..., tret=...          & ! entry
                       ,var = var                    &
                       ,grid = grid                  &
                       )

  IF (RG_STATUS == RGSTAT_STOP) THEN
      WRITE(*,*) 'END OF NAMELIST FILE REACHED !'
      EXIT
  END IF

  IF (CONDITION_ON_VAR(var).OR.CONDITION_ON_GRID(grid)) THEN
      ! REGRID
      WRITE(*,*) 'CONDITION IS TRUE FOR VARIABLE OR GRID !'

      IF (RG_CTRL == RG_SCAN) THEN
         WRITE(*,*) 'REGRIDDING ACCORDING TO CURRENT NAMELIST ...'
         RG_NML  = NML_STAY  ! DO NOT READ NEXT NAMELIST FROM FILE
         RG_CTRL = RG_PROC   ! PERFORM REGRIDDING
      ELSE
         WRITE(*,*) 'REGRIDDING DONE !'
         !!
         !! USE THE REGRIDDED VARIABLES AND THE OUTPUT GRID HERE
         !! POSSIBLY CLEAN 'grid' and 'var'
         !! POSSIBLY EXIT ENDLESS DO LOOP HERE LIKE THIS:
         !RG_CTRL = RG_STOP
         !! this will terminate NCREGRID
         !! (close files, clean memory)
         !! at the next call within the DO-loop
         !!
```

```
            WRITE(*,*) 'SCANNING NEXT NAMELIST ...'
            RG_NML  = NML_NEXT
            RG_CTRL = RG_SCAN
          END IF
      ELSE
          WRITE(*,*) 'CONDITION IS FALSE FOR VARIABLE OR GRID !'
          WRITE(*,*) 'TRYING NEXT NAMELIST ...'
      END IF

  END DO
  ! END REGRIDDING

END SUBROUTINE MY_SUBROUTINE
!-------------------------------------------------------------------------
```

The variable `filename` specifies the file containing the namelist(s) to be read. The default I/O unit for the namelist file is 17. This can be overwritten with the optional integer parameter `iounit`. The parameters `RG_CTRL`, `RG_NML`, and `RG_STATUS` are predefined integer parameters, used to control the regridding procedure:

- `RG_CTRL` switches the regridding procedure. Set to
  - `RG_SCAN`, no regridding is performed; the variable(s) and the grid are read from the *infile* as specified in the namelist (see section 8) and returned in `var` and `grid`, respectively.
  - `RG_PROC`, the regridding procedure is switched on, and the resulting variable(s) and grid are returned in `var` and `grid`, respectively.
  - `RG_STOP`, NCREGRID is terminated (closing files, cleaning memory).
- `RG_NML` controls the namelist file handling. Set to
  - `NML_NEXT`, the next namelist is read from the namelist file before scanning / regridding.
  - `NML_STAY`, the current namelist is used again for scanning / regridding.
- `RG_STATUS` is returned by the subroutine. It is
  - `RGSTAT_START`, if the namelist file (`filename`) was just opened and the first namelist read.
  - `RGSTAT_CONT`, as long as the end of the namelist file is not reached.
  - `RGSTAT_STOP`, if the end of the namelist file has been reached, or `RG_CTRL` has been set to `RG_STOP`.

With the optional logical parameters `lrgx`, `lrgy`, and `lrgz` set to `.FALSE.`, regridding in longitudinal, latitudinal, and vertical direction, respectively, can be switched off, separately. The default for all three parameters is `.TRUE.`.

NCREGRID internally loops over the time-steps of the input netCDF file (*infile*) as specified by the namelist entry `i_t` (see section 8). This can be overwritten by the optional integer parameters `tmin`, `tmax`, `tstep`, and `tret`. The loop is then from `tmin` to `tmax` with a step-size of `tstep`. The grid and variable(s) at position `tret` are returned in `grid` and `var`, respectively. Table 2 gives a complete list of the parameters of the `SUBROUTINE REGRID_CONTROL` parameters.

9.3 Level 2 interface for calling NCREGRID

To facilitate the use of NCREGRID in interface mode, the module `messy_ncregrid_tools.f90` in the subdirectory `./examples/tools` provides some additional high level (level 2) interface routines.

9.3.1 SUBROUTINE RGTOOL_READ_NCVAR

This subroutine reads / regrids a single netCDF-variable (`TYPE ncvar`, see section 10.3) according to a given namelist. The parameters are

```
SUBROUTINE RGTOOL_READ_NCVAR(iou, nmlfile, vname, t, var       &
                            ,grid, lrg, lrgx, lrgy, lrgz, lok)

INTEGER, INTENT(IN)             :: iou          ! logical I/O unit
CHARACTER(LEN=*), INTENT(IN)    :: nmlfile      ! namelist file
CHARACTER(LEN=*), INTENT(IN)    :: vname        ! variable name
INTEGER, INTENT(IN)             :: t            ! netCDF time step
TYPE(ncvar)                     :: var          ! nc-variable
TYPE (geohybgrid),    OPTIONAL  :: grid         ! output grid info
LOGICAL, INTENT(IN),  OPTIONAL  :: lrg          ! regrid really ?
LOGICAL, INTENT(IN),  OPTIONAL  :: lrgx         ! regrid in x
LOGICAL, INTENT(IN),  OPTIONAL  :: lrgy         ! regrid in y
LOGICAL, INTENT(IN),  OPTIONAL  :: lrgz         ! regrid in z
LOGICAL, INTENT(OUT), OPTIONAL  :: lok          ! OK?
```

- `iou` is the I/O unit for opening the namelist file.
- `nmlfile` is the name of the file which contains the regridding namelist.
- `vname` is the name of the variable (cf. section 8.2).
- `t` is the step along the time dimension of `vname`.
- `var` is the netCDF-variable output.
- `grid` is the optional output of the grid-structure of `var`.
- `lrg` is an optional switch for the regridding process, which is `.TRUE.` (default) for regridding, and `.FALSE.` for import of the data on the input grid.

NAME	TYPE	REQ/OPT	DEFAULT	IN/OUT	DESCRIPTION
RG_CTRL	INTEGER	REQ		IN	regrid control flag
RG_NML	INTEGER	REQ		IN	namelist control flag
RG_STAT	INTEGER	REQ		OUT	regrid status flag
nmlfile	CHARACTER(LEN=*)	REQ		IN	namelist file
iounit	INTEGER	OPT	17	IN	I/O unit
lrgx	LOGICAL	OPT	TRUE	IN	lon. regridding ON/OFF
lrgy	LOGICAL	OPT	TRUE	IN	lat. regridding ON/OFF
lrgz	LOGICAL	OPT	TRUE	IN	vert. regridding ON/OFF
tmin	INTEGER	OPT		IN	start time step
tmax	INTEGER	OPT		IN	stop time step
tstep	INTEGER	OPT		IN	time step size
tret	INTEGER	OPT		IN	data time step
var	TYPE(ncvar), DIMENSION(:), POINTER	REQ		OUT	regridded data
grid	TYPE(geohybgrid)	REQ		OUT	grid structure

Table 2: Parameters of the SUBROUTINE REGRID_CONTROL. OPT indicates optional parameters, REQ indicates required parameters. Input parameters (IN) are not changed by the subroutine, output parameters (OUT) are returned by the subroutine.

- `lrgx` is an optional switch for regridding along the longitudinal dimension (default: `.TRUE.`).
- `lrgy` is an optional switch for regridding along the latitudinal dimension (default: `.TRUE.`).
- `lrgz` is an optional switch for regridding along the vertical dimension (default: `.TRUE.`).
- `lok` is an optional status flag which is `.TRUE.` on success, and `.FALSE.` otherwise (e.g., if the namelist file was not found, or if the variable was not in the *infile*).

9.3.2 SUBROUTINE RGTOOL_READ_NCFILE

This subroutine reads / regrids a complete netCDF-file according to a given namelist. The parameters are

```
SUBROUTINE RGTOOL_READ_NCFILE(iou, nmlfile, fname, t, var        &
                             ,grid, lrg, lrgx, lrgy, lrgz, lok)

INTEGER, INTENT(IN)                :: iou          ! logical I/O unit
CHARACTER(LEN=*), INTENT(IN)       :: nmlfile      ! namelist file
CHARACTER(LEN=*), INTENT(IN)       :: fname        ! filename
INTEGER, INTENT(IN)                :: t            ! netCDF time step
TYPE(ncvar), DIMENSION(:), POINTER :: var          ! nc-variables
TYPE (geohybgrid),    OPTIONAL     :: grid         ! output grid info
LOGICAL, INTENT(IN),  OPTIONAL     :: lrg          ! regrid really ?
LOGICAL, INTENT(IN),  OPTIONAL     :: lrgx         ! regrid in x
LOGICAL, INTENT(IN),  OPTIONAL     :: lrgy         ! regrid in y
LOGICAL, INTENT(IN),  OPTIONAL     :: lrgz         ! regrid in z
LOGICAL, INTENT(OUT), OPTIONAL     :: lok          ! OK?
```

- `iou` is the I/O unit for opening the namelist file
- `nmlfile` is the name of the file which contains the regridding namelist.
- `fname` is the name of the input netCDF file (*infile*, cf. section 8).
- `t` is the step along the time dimension specified in the namelist.
- `var` is the list of netCDF output variables list.
- `grid` is the optional output of the grid-structure of the variables in `var`.
- `lrg` is an optional switch for the regridding process which is `.TRUE.` (default) for regridding, and `.FALSE.` for import of the data on the input grid.
- `lrgx` is an optional switch for regridding along the longitudinal dimension (default: `.TRUE.`).

- `lrgy` is an optional switch for regridding along the latitudinal dimension (default: `.TRUE.`).
- `lrgz` is an optional switch for regridding along the vertical dimension (default: `.TRUE.`).
- `lok` is an optional status flag which is `.TRUE.` on success, and `.FALSE.` otherwise (e.g., if the namelist file was not found, or if the input netCDF file was not found).

9.3.3 SUBROUTINE RGTOOL_CONVERT

This subroutine is used to convert the data of a netCDF-variable (`TYPE ncvar`, see section 10.3) into a 4D array of type REAL. The parameters are

```
SUBROUTINE RGTOOL_CONVERT(var, dat, grid, order)

TYPE (ncvar),      INTENT(IN)           :: var   ! nc-variable
REAL, DIMENSION(:,:,:,:), POINTER       :: dat   ! data array
TYPE (geohybgrid), INTENT(IN)           :: grid  ! grid information
CHARACTER(LEN=4),  INTENT(IN), OPTIONAL :: order ! axis order in data
```

- `var` is the netCDF-variable to be converted.
- `dat` is the 4D array (NULL-pointer !) which carries the result.
- `grid` is the underlying grid-structure of `var`.
- `order` is an optional 4-character string, describing the axis order of `dat`. The default is `'xyzn'`, where `x` denotes longitude, `y` latitude, and `z` the vertical direction. `n` is the parameter-axis, e.g., the index range for IXF-type regridding.

9.3.4 SUBROUTINE RGTOOL_G2C

This subroutine is used to convert the spatial coordinates of a grid-structure (`TYPE geohybgrid`, see section 10.3) into 1D- and 2D-arrays. The parameters are:

```
SUBROUTINE RGTOOL_G2C(grid, hyam, hybm, p0, ps         &
                     ,hyai, hybi                       &
                     ,latm, lonm                       &
                     ,lati, loni, nlat, nlon, nlev)

TYPE (geohybgrid), INTENT(IN)           :: grid
REAL, DIMENSION(:),   POINTER, OPTIONAL :: hyam   ! hybrid-A-coeff.
REAL, DIMENSION(:),   POINTER, OPTIONAL :: hybm   ! hybrid-B-coeff.
REAL, DIMENSION(:),   POINTER, OPTIONAL :: p0     ! reference pressure
REAL, DIMENSION(:,:), POINTER, OPTIONAL :: ps     ! surface pressure
REAL, DIMENSION(:),   POINTER, OPTIONAL :: hyai   ! hybrid-A-coeff.
REAL, DIMENSION(:),   POINTER, OPTIONAL :: hybi   ! hybrid-B-coeff.
REAL, DIMENSION(:),   POINTER, OPTIONAL :: latm   ! latitude
```

```
REAL, DIMENSION(:),   POINTER, OPTIONAL :: lonm   ! longitude
REAL, DIMENSION(:),   POINTER, OPTIONAL :: lati   ! latitude
REAL, DIMENSION(:),   POINTER, OPTIONAL :: loni   ! longitude
INTEGER,           INTENT(IN), OPTIONAL :: nlat
INTEGER,           INTENT(IN), OPTIONAL :: nlon
INTEGER,           INTENT(IN), OPTIONAL :: nlev
```

- `grid` is the grid structure.
- `hyam` is the list of hybrid-A-coefficients (box-mid).
- `hybm` is the list of hybrid-B-coefficients (box-mid).
- `p0` is the reference pressure (1D-array of length 1).
- `ps` is the surface pressure field (2D).
- `hyai` is the list of hybrid-A-coefficients (interfaces).
- `hybi` is the list of hybrid-B-coefficients (interfaces).
- `latm` is the list of latitudes (box-mid).
- `lonm` is the list of longitudes (box-mid).
- `lati` is the list of latitudes (interfaces).
- `loni` is the list of longitudes (interfaces).
- `nlat` is an optional parameter specifying the number of latitudes (box-mid).
- `nlon` is an optional parameter specifying the number of longitudes (box-mid).
- `nlev` is an optional parameter specifying the number of levels (box-mid).

Note: In case the grid is less than 3-dimensional (e.g., latitude × longitude), and the INTEGER parameter/s of the missing dimension (i.e., in the example `nlev`) is/are not specified, the subroutine returns a NULL-pointer for all corresponding arrays (i.e., in the example for `hyam`, `hybm`, `hyai`, and `hybi`). If the parameter/s is/are specified, however, the subroutine returns corresponding array/s of the requested length (in the example `nlev`), filled with 0.0 (i.e., in the example `hyam(1:nlev) = 0.0`, `hybm(1:nlev) = 0.0`, `hyai(1:nlev+1) = 0.0`, `hybi(1:nlev+1) = 0.0`). This works accordingly for the other dimensions.

9.4 Handling counter information (time stepping)

In applications of NCREGRID in interface mode, it might be useful to trigger the reading / regridding procedure of NCREGRID under certain conditions (events) for reading / regridding a defined time-slice of one or more variables from a netCDF-file. For example, one might wish to update a set of gridded boundary conditions once a month during a multi-month model simulation. The file `messy_ncregrid_tools.f90` contains an additional TYPE declaration, and the corresponding subroutines to support this functionality.

9.4.1 TYPE NCRGCNT

This type provides a structure to store counter information:

```
TYPE NCRGCNT
   CHARACTER(LEN=NCCNTMAXNLEN) :: name = ''
   INTEGER                     :: start = 1
   INTEGER                     :: step  = 1
   INTEGER                     :: reset = 1
   INTEGER                     :: current = 1
END TYPE NCRGCNT
```

`name` is used to identify the counter (e.g., in a list of counters), `start` contains the minimum counter position, `step` the increment, and `reset` the maximum counter position. The `current` counter position can for instance be used to read / regrid a specific time-step of a netCDF-variable by setting the parameter `t` of `SUBROUTINE RGTOOL_READ_NCVAR` or `SUBROUTINE RGTOOL_READ_NCFILE` to `current`.

9.4.2 SUBROUTINE RGTOOL_NCRGCNT_RST

This subroutine provides an interface for the automatic handling of counter information, including

- the update of the counter, if it is triggered: The actual counter position `current` is incremented by `step`. If the result is larger than `reset`, `current` is reset to `start`. This allows cyclic counting.
- the unambiguous storage of a specific counter in a central list to be accessible from everywhere, and to be potentially saved in an external file.
- the continuous alignment of the actual counter and its copy stored in the central list.

The subroutine is called with the following parameters:

```
RGTOOL_NCRGCNT_RST(mname, start, restart, event, c, lout, linit)

CHARACTER(LEN=*), INTENT(IN)    :: mname   ! name of the list
LOGICAL,          INTENT(IN)    :: start   ! .TRUE. at first time step
```

```
LOGICAL,          INTENT(IN)    :: restart ! .TRUE. at first time step
                                          ! of restart
LOGICAL,          INTENT(IN)    :: event   ! .TRUE. on event
TYPE(NCRGCNT),    INTENT(INOUT) :: c       ! counter - struct
LOGICAL,          INTENT(IN)           :: lout
LOGICAL,          INTENT(IN), OPTIONAL :: linit
```

- `mname` is a name identifying a sub-list of counters in the central list.
- `start` must be `.TRUE.` only at the very first step. It is used to prevent a counter update immediately at start, if at the same time the `event` is triggered.
- `restart` must be `.TRUE.` only after the first initialisation of the counter list from an external file. This is used to restore the actual counter from the list, even if the `event` is not triggered.
- `event` is `.TRUE.`, indicating that the counter update is triggered.
- `c` is the counter structure of the actual counter.
- `lout` is `.TRUE.` for diagnostic output.
- `linit` is an optional switch (default: `.FALSE.`). If set to `.TRUE.`, it prevents counters of the same name and `mname` to be updated (counter exists already in the list), but creates those which are new.

The sequence of operations is as follows:

1. If `start`, `restart`, and `event` are all `.FALSE.`, the subroutine is left immediately.
2. The algorithm locates the specified counter `c` by its name in the sub-list `mname`. If the counter is found, the actual counter `c` is restored from its list entry.
3. The initialisation flag `linit` is checked: Only if `linit` is `.TRUE.` the presence of the counter in the list is re-evaluated: If it exists in the list, the routine stops with an error, since during the initialisation, the counter must not yet exist. If the counter is new, however, it is saved in the list.
4. The algorithm checks, if the update has been triggered. Only if `event` is `.TRUE.` and `start` is `.FALSE.`, an existing actual counter and its copy in the central list are updated, as described above. If the counter does not exist in the list, the routine terminates with an error.

9.4.3 More on counter information

With the SUBROUTINE WRITE_NCRGCNT_LIST the complete central list of currently stored counter information can be dumped into an ASCII file. Likewise, with SUBROUTINE READ_NCRGCNT_LIST, the counter information can be restored from a file previously written by SUBROUTINE WRITE_NCRGCNT_LIST. This functionality is required, if counter information must be saved for later usage, as for instance, if a model simulation is broken down into a chain simulation, whereby each chain element must start exactly where the previous one ended.

The SUBROUTINE GET_NEXT_NCRGCNT can be used to loop over all currently stored counters in the internal list, e.g., to search for a specific counter:

```
LOGICAL,                :: last
TYPE(ncrgcnt), POINTER :: cptr
...
DO
    CALL GET_NEXT_NCRGCNT(last, cptr)
    IF (last) EXIT
    ! check cptr here
END DO
```

last is .TRUE., if the end of the list is reached. This indirect construction is required, since the number of counters stored in the internal list is not known a priori.

And last, but not least, SUBROUTINE CLEAN_NCRGCNT_LIST is used to remove all counter list entries, i.e., to clean up the memory.

10 Behind the Scenes of NCREGRID

10.1 Overview of the different modules

NCREGRID is hierarchically organised in a number of modules, as shown in Figure 1.

10.2 Sequence of operations

Figure 2 shows the steps between reading a namelist and the resulting output. During the process, input and output grid are

- checked for consistency,
- reordered to achieve monotonous axes for NREGRID,
- completed (i.e., box-mids are calculated from interfaces, or vice-versa).

Furthermore, the grid-information of the input (*infile*) and output grid (*grdfile* or INTERFACE_GEOHYBGRID) is balanced, such that everything which is not specified for the output grid is taken from the input grid. This allows, e.g., a horizontal regridding

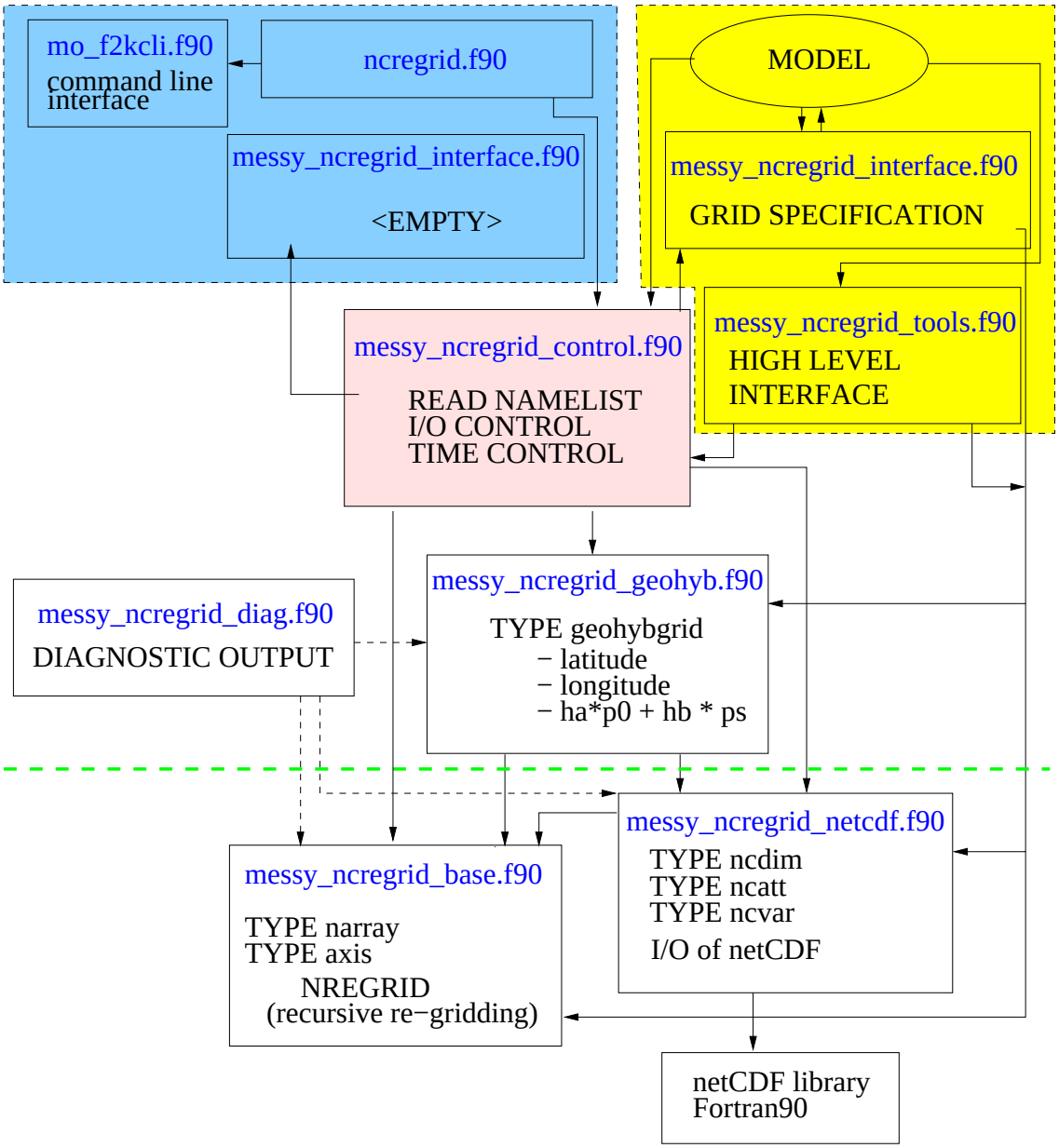

Figure 1: Overview of the different NCREGRID modules with TYPE declarations and dependencies. The blue shaded parts are only needed for the stand-alone mode. The yellow shaded areas are required for the coupling to a model. Modules below the green dashed line are independent on the special requirements of geo-hybrid grids, and can be applied to different grid-types.

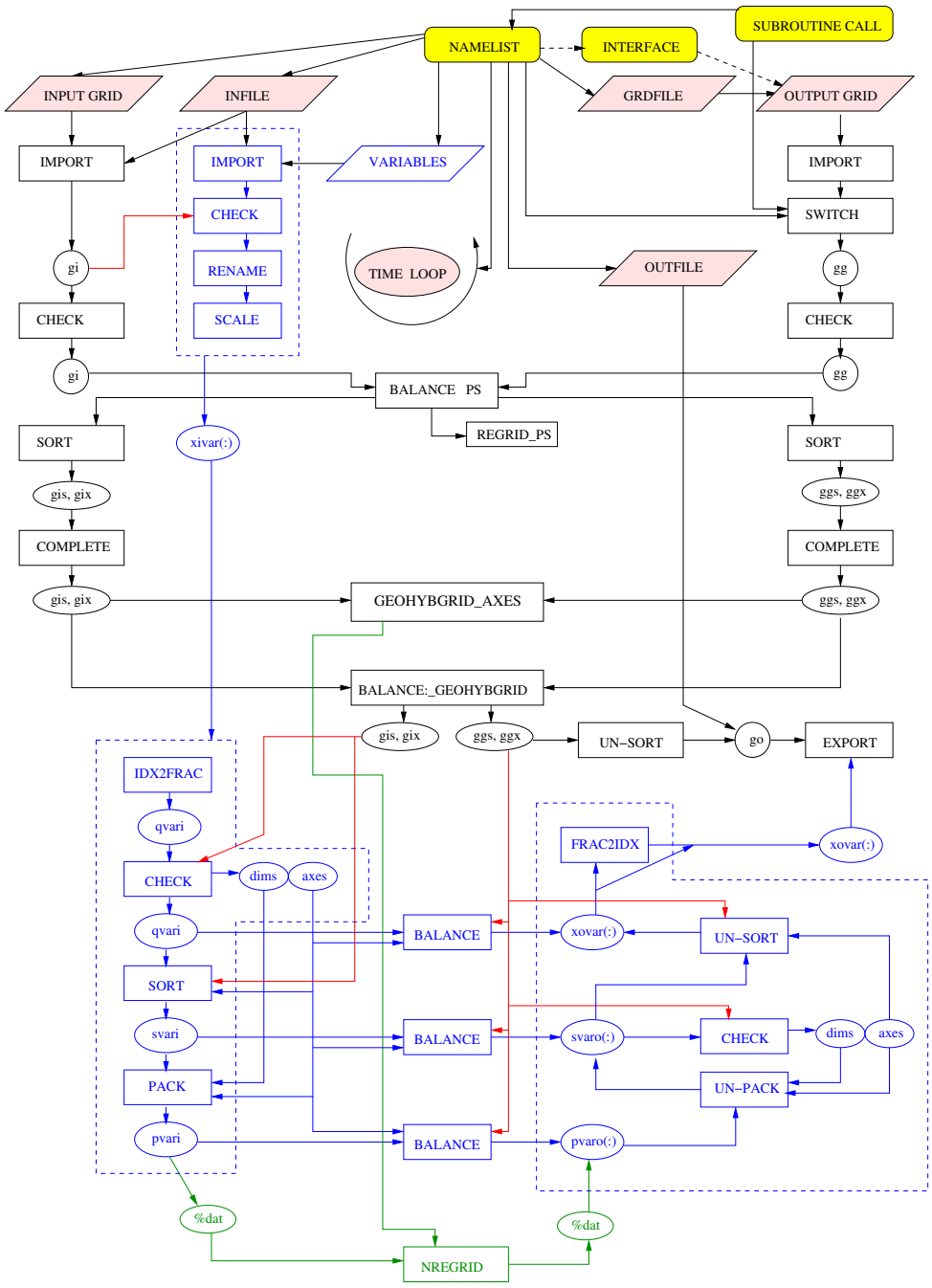

Figure 2: Sequence of operations in NCREGRID. Boxes show operations, circles / ellipses show information stored in structures (given are the used Fortran95 variable names). Dashed boxes indicate operations within a loop over variables. Black lines are used for operations on grid structures, blue lines are used for operations on data variables, and red lines indicate connections between grid structures and data variables. Shown in green is the core regridding step. The yellow shaded boxes present the three possibilities for user interaction (The grid specification via INTERFACE and the options of SUBROUTINE CALL are only accessible in the interface mode of NCREGRID). Finally, the red boxes show the basic information derived from the namelist.

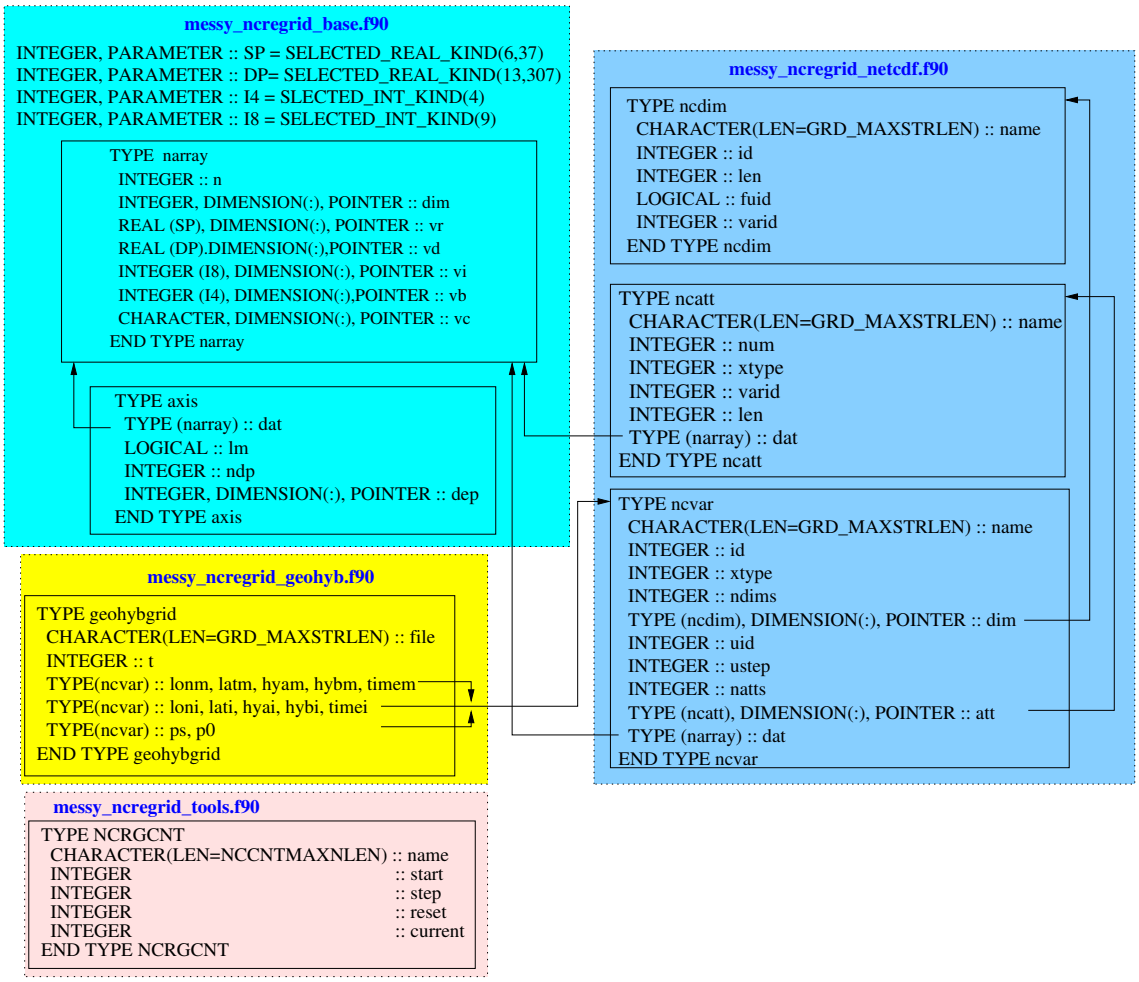

Figure 3: Type definitions (Fortran95 structures) of the different modules and their dependencies.

of 3D (spatial) scalar fields, while conserving the vertical hybrid-coordinate system of the source file. In a similar way, only vertical regridding can be performed.

Furthermore, if the surface pressure of the output grid is not available (or should not be used!), the input surface pressure is pre-regridded and used for the output grid. Similarly, the output surface pressure (in *grdfile*) is used, if the input surface pressure is omitted (`BALANCE_GEOHYBGRID_PS`).

This balancing allows a flexible handling of the data for many purposes.

10.3 TYPE definitions

Figure 3 shows in detail the various TYPE definitions in the different modules and their inter-connections:

- `TYPE narray` (`messy_ncregrid_base.f90`) provides a container for data fields of arbitrary rank N (N-dimensional data) and type (float, double precision, integer, byte, character). `n` is the rank N of the `narray`, i.e., the number

of dimensions, and dim an integer vector of length n containing the dimension length(s). The pointers point to the memory containing the field data, organised as a 1-dimensional array. The module messy_ncregrid_base.f90 provides the

```
INTEGER FUNCTION POSITION(dim, vec)
```

which calculates for the element with integer position vector vec(:) the position m in the linear data array, if the array is interpreted as an N-dimensional array with dimension length(s) dim(:). The reverse procedure is provided by the

```
SUBROUTINE ELEMENT(dim, m, vec)
```

This construction is required, since in Fortran95 the rank of an array cannot be defined at runtime.

- TYPE axis (messy_ncregrid_base.f90) contains the discrete values of an arbitrary axis. The logical lm is .TRUE. for modulo-axes. Further, ndp is the number of dependencies, which is 1 for independent axes (such as longitude and latitude), and > 0 for curvilinear axes (such as, e.g., pressure levels, which depend for hybrid-grids on latitude, longitude, (and time)). The integer vector dep(:) contains the numbers (indices) of the dependent axes, if the axes are combined in a list of

  ```
  TYPE (axis), DIMENSION(:), ALLOCATABLE / POINTER
  ```

- TYPE ncdim, TYPE ncvar, and TYPE ncatt (messy_ncregrid_nc.f90) provide three Fortran95 structures for handling the contents of a netCDF file (see netCDF manual), namely dimensions, variables, and attributes, respectively. This module also contains powerful, robust routines for netCDF input / output into / from these structures, such as

  ```
  SUBROUTINE IMPORT_NCDIM(...)
  SUBROUTINE EXPORT_NCDIM(...)
  SUBROUTINE IMPORT_NCATT(...)
  SUBROUTINE EXPORT_NCATT(...)
  SUBROUTINE IMPORT_NCVAR(...)
  SUBROUTINE EXPORT_NCVAR(...)
  ```

- TYPE geohybgrid (messy_ncregrid_geohyb.f90) defines the geo-hybrid-grid structure, using netCDF variables (TYPE ncvar) for the grid box mid points (...m) and the grid box interfaces (...i). t is the time-step of the current grid, and file the netCDF filename for I/O.

- `TYPE ncrgcnt` (`messy_ncregrid_tools.f90` provides a structure for the cyclic counting, as described in section 9.4.

10.4 The recursive core of NCREGRID

The actual regridding is performed by the

```
RECURSIVE SUBROUTINE NREGRID(...)
```

in module `messy_ncregrid_base.f90`. The algorithm has to be recursive to allow the regridding in an arbitrary number of dimensions.

On the first recursion level, some basic checks on the input are performed. NREGRID compares the input / output axes and specifies a dimension as invariant, if the output axis is undefined. For invariant dimensions, no regridding is required. If a dimension is invariant and the respective axis is dependent (i.e., curvilinear), the axis data is pre-regridded. The whole regridding procedure requires a special order of the dimensions, which is calculated by NREGRID itself (on the first recursion level): first, all independent, non-invariant dimensions, next all dependent, non-invariant dimensions, and finally all invariant dimensions.

If the current dimension is non-invariant, NREGRID calculates the overlap - matrices (as fractional overlap) between source and destination axis. For each non-zero element of these matrices, it branches to the next recursion level (i.e., to the next dimension).

If the current dimension is the first invariant dimension, NREGRID loops along all remaining (invariant) dimensions and calculates the results for the output grid, depending on the regridding type (RG_TYPE, see section 7).

Since in most practical cases, regridding is performed for more than one variable on the same grid (axes), NREGRID handles a list of variables in one step. The innermost loop is over the number of variables, in oder to increase the performance, i.e., to avoid recalculation of the same overlap-matrices for all variables separately.

Acknowledgements

- Rolf Sander (MPI for Chemistry, Mainz) for implementation in ECHAM-5, testing the code, and comments on this documentation
- Philip Stier (MPI for Meteorology, Hamburg) for implementation in ECHAM-5 and testing the code
- Huisheng Bian (Dept. of Earth System Science, Univ. of California, Irvine) for valuable bug-reports of the version 0.8b
- Benedikt Steil (MPI for Chemistry, Mainz) for valuable bug reports
- Pascal Terray (Laboratoire d'Océanographie Dynamique et de Climatologie) for reporting the successful installation on SGI
- Frank Peacock for reporting the successful installation on Windows 98
- Maarten van Aalst for the successful installation on SGI, IRIX 6.5
- Norman Wood (Colorado State University) for pointing out the Fortran95 extensions used in the code
- Ingo Kirchner (FU-Berlin) for the successful installation on PC-Linux with the NAG compiler
- Mike Bauer (NASA GISS) for the successful compilation on Apple G5 (OS v10.4.4 with xlf90 v8.1)