ECHAM5/MESSy (Version 0.9) User Manual

Patrick Jöckel & Rolf Sander

joeckel@mpch-mainz.mpg.de, sander@mpch-mainz.mpg.de
Air Chemistry Department
Max-Planck Institute of Chemistry
PO Box 3060, 55020 Mainz, Germany
www.messy-interface.org

Version from January 31, 2005

Preamble

"When you work on a software project, one of your short-term goals is to solve a problem at hand. If you are doing this because someone asked you to solve the problem, then all you need to do to look good in per eyes is to deliver a program that works. Nevertheless, regardless of how little person may appreciate this, doing just that is not good enough. Once you have code that gives the right answer to a specific set of problems, you will want to make improvements to it. As you make these improvements, you would like to have proof that your code's known reliability hasn't regressed. Also, tomorrow you will want to move on to a different set of related problems by repeating as little work as possible. Finally, one day you may want to pass the project on to someone else or recruit another developer to help you out with certain parts. You need to make it possible for the other person to get up to speed without reinventing your efforts. To accomplish these equally important goals you need to write good code.

...

Please do not hold the opinion that contributions in science and engineering are 'true' contributions and software development is just a 'tool'. This bigotted attitude is behind the thousands of lines of ugly unmaintainable code that goes around in many places. Good software development can be an important contribution in its own right, and regardless of what your goals are, please appreciate it and encourage it." (http://autotoolset.sourceforge.net/tutorial.html)

A very useful document is available from the UK Met Office (UKMO) at http://www.meto.govt.uk/research/nwp/numerical/fortran90/f90_standards.html. The standards described for MESSy and in the UKMO document are not just "technical" and "cosmetical" contributions to the code. They are of prime importance to develop a well documented code that is easy to understand and therefore also easy to debug if necessary. Ignoring the conventions or delaying their implementation would slow down any further development, the evaluation phase, and the debugging-phase of the model setup. Therefore, code won't be accepted for the official MESSy-versions unless it fulfills our conventions.

Contents

1 Introduction

The highly structured Modular Earth Submodel System (MESSy) is used to couple several submodels to the general circulation model (GCM) ECHAM5 (Roeckner et al., The atmospheric general circulation model ECHAM 5. PART I: Model description, MPI-Report, 349, 2003 (`http://www.mpimet.mpg.de/en/extra/models/echam/mpi_report349.pdf`)), thus extending it into a fully coupled chemistry-climate model. ECHAM5/MESSy is implemented to run in parallel using the Message Passing Interface (MPI, `http://www-unix.mcs.anl.gov/mpi/index.html`, `http://www-unix.mcs.anl.gov/mpi/mpich/`).

Information on the latest development status of MESSy and the ECHAM5/MESSy activities (including updates of this manual), are available on the MESSy web-pages (`http://www.messy-interface.org`). Furthermore, it is recommended to read the files `README_MESSY`, `CHANGELOG`, and `WARNINGS` of the ECHAM5/MESSy distribution.

This user manual supplies basic information on the structure of the ECHAM5/MESSy code and on how to use the comprehensive model setup. Further information on specific submodels can be found on the MESSy web-pages. If this is not sufficient, contact the submodel maintainers.

For questions, bug-reports, etc., with respect to the ECHAM5/MESSy setup, please contact the mailing address `messy@mpch-mainz.mpg.de`.

Section 2 describes in detail the model structure of ECHAM5/MESSy and its components. Understanding its contents is a prerequisite for modifying any model code or writing an additional submodel. However, users who only want to run the model can skip it and go directly to sections 3. and 4. Submodel developers, who want to contribute their code also need to read section 5.

2 The ECHAM5/MESSy model structure

2.1 Implementation of the MESSy interface

The general description of the four layer MESSy interface structure and the MESSy coding conventions can be found in Jöckel et al. [2005]. Here, the specific implementation of MESSy for the GCM ECHAM5 is described. A detailed overview of the implementation can be found in the accompanying ECHAM5/MESSy reference card (`messy_echam5_ref.pdf`).

2.1.1 The base model layer (BML)

The GCM ECHAM5 serves as the base model, i.e., constitutes the base model layer (BML) of the MESSy implementation. Information about the original ECHAM5 can be found on the web-page (`http://www.mpimet.mpg.de/en/extra/models/echam/echam5.php`), which is also accessible via the MESSy web-pages. Additional modifications / extensions of the base model ECHAM5 are also documented on the MESSy web-pages.

The main entry points of MESSy in ECHAM5 are shown in Figure 1, which gives a simplified overview of the upper three MESSy layers depicted as a flow diagram. According to the MESSy standard, the entry points in ECHAM5 are encapsulated in preprocessor directives:

```
#ifdef MESSY
  CALL messy_<...>
#endif
```

The specific role of these main entry points is explained in section 2.2 below.

2.1.2 The base model interface layer (BMIL)

The MESSy base model interface layer comprises the following files:

- central submodel control interface:
  - `messy_main_switch.f90` is the SMCL of the generic submodel main_switch. It provides one global switch for each individual submodel. These switches are imported from the `CTRL`-namelist in the file `MESSy.nml`. `MESSy.nml` is written by the run-script `xmessy` where the user selects the submodels to be activated (user interface, see section 2.1.5).
  - `messy_main_switch_e5.f90` is the SMIL of the generic submodel main_switch. It provides the `PUBLIC` subroutine for the overall MESSy initialization called by `messy_main_control_e5.f90`. Here, the distribution of the submodel switches to the different processes in a parallel environment is performed.
  - `messy_main_control_e5.f90` (Figure 1) is the SMIL of the generic submodel main_control. It provides the subroutines for the main entry points from ECHAM5 into MESSy. The individual MESSy submodels are called from here. A specific submodel is only called if it has been activated (user interface, see section 2.1.5). Table 1 shows in detail how these MESSy main entry points are embedded in the call tree of ECHAM5.
- data transfer/export interface: Data export of results and data dumping for the restart handling in chain experiments are currently based on the ECHAM5 stream implementation (see section 2.3). In particular:
  - `messy_main_tracer.f90` is the SMCL of the generic submodel main_tracer. It constitutes a completely new, base model independent implementation (different from that in the original ECHAM5 !) for handling of chemical compounds in arbitrary representations (see sections 2.3 and 2.4).

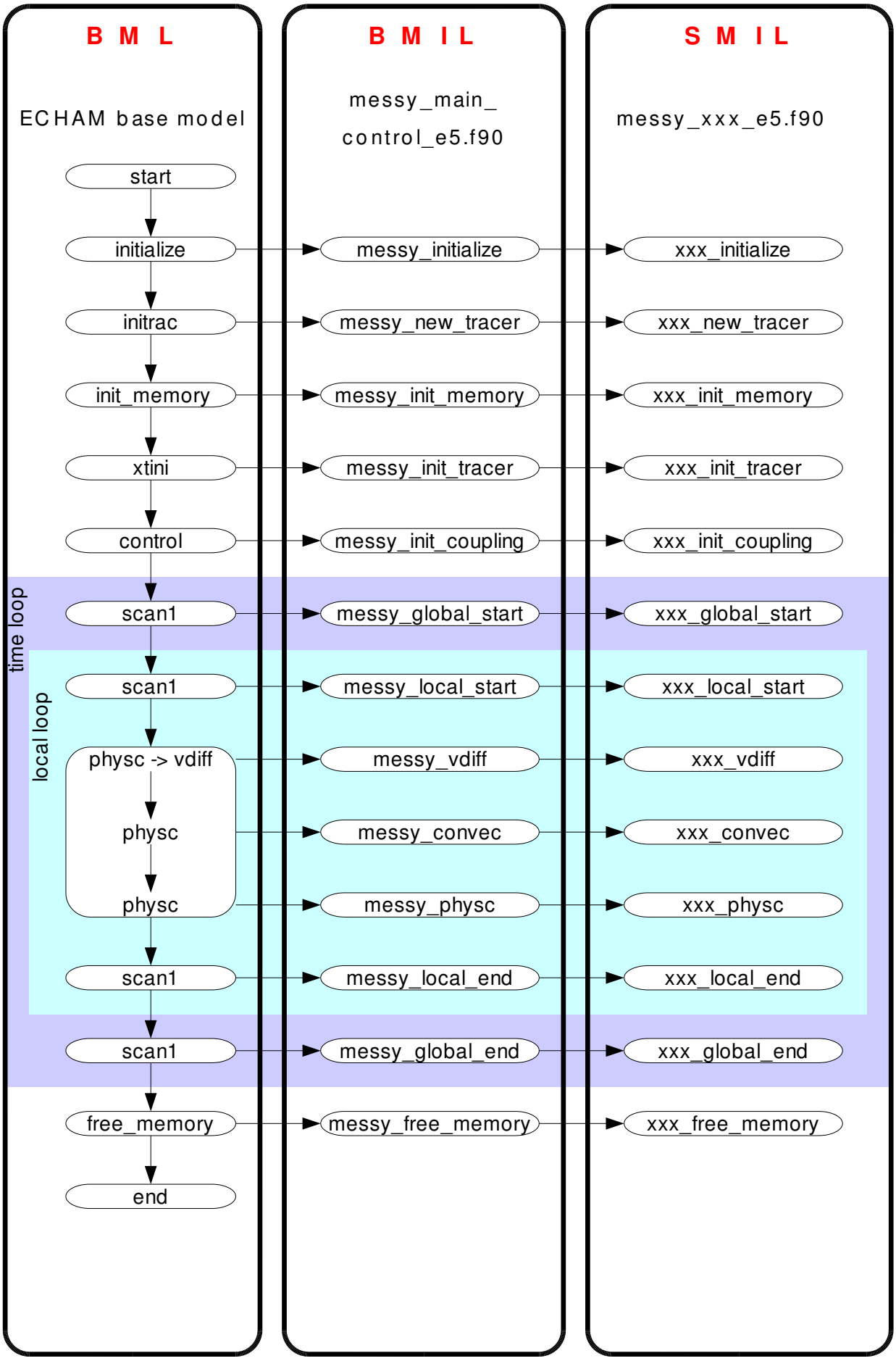
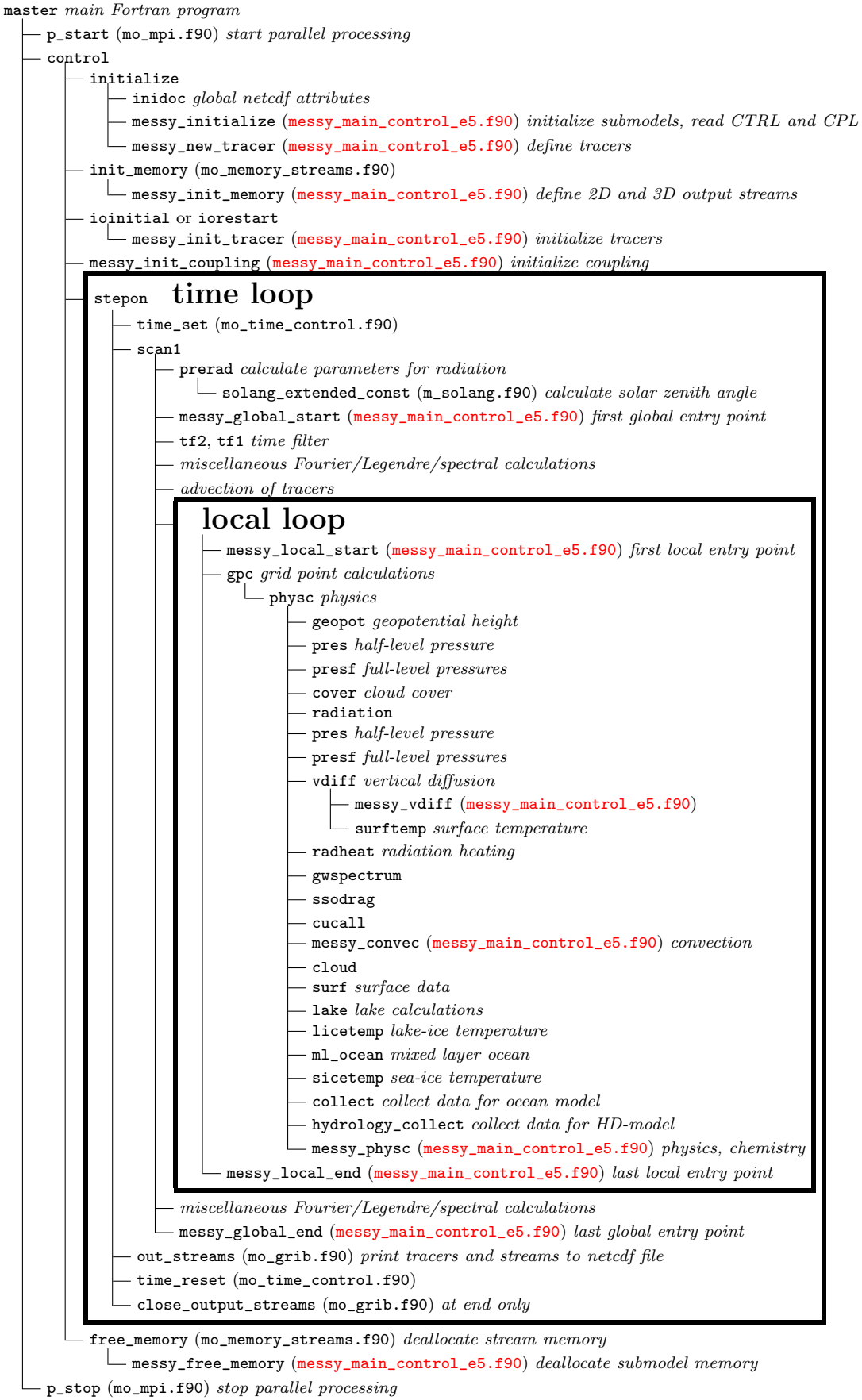


Figure 1: Main entry points of MESSy in ECHAM5 shown for an arbitrary submodel called xxx.

Table 1: ECHAM5/MESSy call tree showing how the MESSy main control interface is embedded into ECHAM5.



  - messy_main_tracer_e5.f90 is the corresponding SMIL, providing the base model dependent setup, in particular the link to the data transfer and export interface (currently the ECHAM5 stream implementation, see section 2.3) for flexible data handling in parallel environments and for data export. Currently, tracer sets in GRIDPOINT and LAGRANGIAN representation are used (see section 2.4).
  - messy_main_data_e5.f90 is used for the two way exchange of physical fields between the base model (i.e., the GCM) and the submodels. It is directly based on the data transfer and export interface (currently the ECHAM5 stream implementation, see section 2.3).
  - messy_main_constants_mem.f90 provides basic physical constants and machine precision constants (KIND-parameters). Note that this file is a memory file which contains neither functions nor subroutines.

- data import interface: The data import interface is based on the rediscretization tool NCREGRID (http://www.mpch-mainz.mpg.de/~joeckel/ncregrid) using netCDF files (http://www.unidata.ucar.edu/packages/netcdf/) as input/output. The coupling to ECHAM5 is performed by the following files:
  - messy_ncregrid_interface.f90 provides the grid specification (spatial resolution) from the base model (ECHAM5) to NCREGRID (the core for the rediscretization of gridded geoscientific data).
  - messy_ncregrid_tools_e5.f90 provides standardized subroutines (i.e., a front end) which can be used by the different submodels to import external, gridded data from netCDF files.
- messy_main_tools.f90 and messy_main_tools_e5.f90 provide the SMCL and the SMIL of an additional generic submodel (toolbox), which contains subroutines and functions (base model independent and dependent, respectively) shared by different submodels.

2.1.3 The submodel interface layer (SMIL)

The SMIL of a specific submodel named <submodel> comprises the following files and serves the following purposes:

- messy_<submodel>_e5.f90 is the name of the submodel interface file (SMIL), which is located in the messy/e5 subdirectory. This file provides the PUBLIC subroutines that are called from the central management control interface (messy_main_control_e5.f90). The MESSy naming convention for the subroutines is shown in Figure 1 for an arbitrary submodel named xxx. The subroutines in this file must in general not contain any parameters. Data transfer to/from the SMCL of the specific submodel is preferably performed via parameters of the SMCL subroutines / functions. As an alternative, PUBLIC variables defined in the SMCL files can be accessed via the Fortran95 USE statement.
- <submodel>_t.nml is a namelist file located in the messy/nml subdirectory and is part of the user interface (see section 2.1.5), controlling the initialization of chemical species (= tracers) at the beginning of a model run, using the data import interface.
- <submodel>.nml (see also sections 2.1.4 and 2.1.5) is a namelist file located in the messy/nml subdirectory and part of the user interface (see section 2.1.5). Within the SMIL it performs two tasks:
  - It contains the &CPL namelist, which holds the specifications for coupling the submodel to the base model and to other submodels.
  - It contains further namelists (&RGTEVENTS and &REGRID) for time dependent gridded boundary condition import via the data import interface (see NCREGRID documentation and the MESSy web-pages for further details).

2.1.4 The submodel core layer (SMCL)

The SMCL of a specific submodel named <submodel> comprises the following files and serves the following purposes:

- messy_<submodel>.f90 is the name of the submodel core file (SMCL), which is located in the messy/src subdirectory. This file provides PUBLIC subroutines which are called by the SMIL of the submodel (messy_<submodel>_e5.f90). It further comprises PRIVATE subroutines which are only used internally. Data transfer to/from the SMIL of the specific submodel is preferably performed via parameters of the PUBLIC subroutines / functions. As an alternative, the SMIL may alter PUBLIC variables defined in the SMCL which have been included into the SMIL via the Fortran95 USE statement. Moreover, this file follows these conventions:
  - It provides a public module identification string modstr and a version modver:

```
CHARACTER(LEN=*), PUBLIC, PARAMETER :: &
   modstr = '<submodel>'
CHARACTER(LEN=*), PUBLIC, PARAMETER :: &
   modver = '<version>'
```

  - The SMCL routines must be resolution independent, independent of the base model grid geometry, independent of the parallel environment, and independent of other submodels.
  - STOP statements must be avoided in the code due to parallel processing. Using a STOP statement in a parallel environment (e.g., MPI) would only terminate one process but not on all of them. Instead of STOP statements, the subroutines of the SMCL must return a status flag (INTEGER), which is 0, if no error occurs. This status flag

can be used by the interface routines (SMIL) to write information to the output, and/or to terminate the base model in case of severe errors.

- In parallel environments, such as MPI, input/output (i.e. `READ`-, `WRITE`-, or `PRINT`-statements) must generally occur only in those SMCL routines, which are exclusively executed by a dedicated I/O-process. The better way is to perform this kind of I/O within the SMIL, where the parallel environment is known.

- `messy_<submodel>_*.f90` according to the general MESSy filename convention (e.g., memory files) are used if more than one SMCL file is required (e.g., in case of sub-submodels, for the sake of clearness, etc.). All conventions for `messy_<submodel>.f90` apply to these files accordingly (example: `messy_ncregrid_*.f90`)

- `<submodel>.nml` (see also sections 2.1.3 and 2.1.5) is a namelist file located in the `messy/nml` subdirectory and is part of the user interface. For the SMCL, it contains the `&CTRL` namelist, which holds all switches and parameters for controlling the internal flow and complexity of the submodel. It might further contain specific additional namelists used by the SMCL of the submodel.

2.1.5 The user interface

The user controls the model run including the submodel activities (i.e., the executable) via the user interface which is implemented using the Fortran95 namelist constructs and a generalized run-script. An overview of the involved files is outlined:

- `xmessy` is the general run-script (sh-script). This script incorporates several tasks which require user interaction:

  - `xmessy` can be run in the foreground, background, and be submitted to a job scheduling system (queuing system). In the latter case, the script automatically detects the type of the scheduling system. Specific resources requested from the used queuing system, e.g., number of CPUs, time limit, log-file redirection etc., must be specified by the user in the header of `xmessy`.
  - Directories for data input (initialization files, files containing gridded, temporal changing boundary conditions, etc., see section 2.7) and a working directory (for output of model results and intermediate files) can be set by the user, if the (host specific) default is not applicable.
  - Some general settings for the base model (e.g., the horizontal and vertical resolution, etc.) are chosen here. These specifications are passed to the namelist file which is controlling the base model (i.e., the GCM ECHAM5). The namelist file is selected with the shell-variable `NML_ECHAM` (see below).
  - The submodel switches (for user interaction) are located here. To use a specific submodel, it must be explicitely activated by

    ```
    USE_<submodel>=T
    ```

    If this line is commented out, the submodel is deactivated. Switches for all submodels are written by `xmessy` to the namelist file `MESSy.nml`.
  - The namelist files for controlling a specific submodel are selected here, one for internal and coupling control, and one for tracer initialization (if applicable):

    ```
    NML_<submodel>=...
    NML_<submodel>_T=...
    ```

    The filenames on the right hand side are arbitrary; they are copied by `xmessy` to `<submodel>.nml` and `<submodel>_t.nml` in the working directory, respectively. This namelist file selection has only an effect, if the corresponding submodel has been activated. The possibility to select from different namelist files allows keeping specific setups (e.g., model experiments) in parallel. In the same way, also the base model namelist file can be selected

    ```
    NML_ECHAM=...
    ```

    which is copied by `xmessy` to `ECHAM5.nml`.

  Furthermore, `xmessy` contains an automatic restart-facility for running job chains. More information on `xmessy` can be obtained by running `xmessy -h`.

- `ECHAM5.nml` is the namelist file controlling the base model. Which namelist file is used, is selected in the run-script (see above).

- `MESSy.nml` is the namelist file for switching the submodels on/off. This file is automatically written by the run-script and does not require any user interaction.

- `<submodel>.nml` is the central namelist file for controlling a specific submodel. It contains the `&CTRL` namelist for operating the internal (i.e., the base model independent) complexity of the submodel, the `&CPL` namelist for managing the coupling between the submodel and the base model, and to other submodels, and further namelists for controlling the import of time dependent gridded boundary conditions via the data import interface (i.e., `&RGTEVENTS` and `&REGRID` namelists). Moreover, it can contain additional submodel specific namelists used in the SMCL. In case a submodel incorporates one or more sub-submodel(s), the namelists of the sub-submodel(s) are called `&CTRL_<sub-submodel>` and `&CPL_<sub-submodel>`, respectively.

- `<submodel>_t.nml` is the submodel specific namelist file controlling the tracer initialization at the beginning of a model run. Import of initial tracer data is performed via the data import interface. For this, the file contains one or more `&REGRID` namelists for NCREGRID (see NCREGRID documentation).

Within the namelist files, variables (beginning with $) are automatically replaced by the equivalent shell variables of `xmessy`, e.g., the input paths (i.e., the shell variables `INPUTDIR_MESSY` and `INPUTDIR_ECHAM5_INI`, see section 2.7).

2.2 The ECHAM5/MESSy call tree

An ECHAM5/MESSy model run can be subdivided into three phases: the initialization phase, the time integration phase (time loop), and the finalizing phase (cleanup). MESSy provides several entry points at which the submodels interact with the base model ECHAM5 as shown in Figure 1.

2.2.1 The initialization phase

The initialization phase comprises 5 entry points:

- `messy_initialize`
  - first initializes the generic submodel main_switch, i.e., the switches (`USE_<submodel>`) of activated submodels are set to `.true.`, by importing the `&CTRL` namelist from `MESSy.nml`.
  - subsequently calls all initialization routines of the activated submodels. In `<submodel>_initialize`, both, the `&CTRL` and `&CPL` namelist of an activated submodel are read. This is performed by calling the SMCL subroutine `<submodel>_read_nml` (in `messy_<submodel>.f90`) and the SMIL subroutine `<submodel>_read_nml_e5` (`PRIVATE`, in `messy_<submodel>_e5.f90`), respectively. These calls are performed only by a dedicated I/O-process in the parallel environment. All namelist entries are then distributed to the different processes in `<submodel>_initialize`.
- `messy_new_tracer`
  - first initializes the generic submodel main_tracer in order to provide the framework for subsequent definition of chemical species by the activated submodels (see section 2.4.1)
  - provides the main entry point for the submodels to define chemical species (see section 2.4.2).
  - at the end sets up the memory and information structures for all tracers in all tracer sets (second part of initializing the generic submodel main_tracer).
- `messy_init_memory`
  - initializes the generic submodel main_data for data exchange between the submodels and the base model
  - links the tracer memory to the data transfer / exchange interface (i.e., currently to the ECHAM5 stream facility, see section 2.3)
  - provides the main entry point for the definition of submodel specific data objects (currently stream elements, see section 2.3) and the allocation of local memory, which can or must not be represented by data objects
- `messy_init_tracer`
  - handles the tracer initialization (see section 2.4.3)
  - provides the main entry point for submodel specific tracer initializations. The easiest way to perform a submodel specific tracer initialization, is by calling the subroutine `tracer_init(modstr)` of the generic submodel main_tracer (`messy_main_tracer_e5.f90`) from the SMIL subroutine `<submodel>_init_tracer`. Here, `modstr= <submodel>` is the identification string of the submodel (see section 2.1.4). Tracer fields are then imported via the data import interface by processing the namelist file `<submodel>_t.nml`.
- `messy_init_coupling`
  - prepares the coupling to other submodels and to the base model using the information that was imported from the `&CPL` namelist in `messy_initialize`.

2.2.2 The time integration phase

The time integration phase (time loop) comprises 7 entry points. At the first and last (`messy_global_start` and `messy_global_end`) the activated submodels have access to the entire data fields after decomposition on the different processes in the parallel environment. Note that this is different from access to the global data without decomposition! The other 5 entry points (`messy_local_start`, `messy_vdiff`, `messy_convec`, `messy_physc`, and `messy_local_end`) are within the local loop. The local loop takes care of an efficient vectorization and parallelization. The drawback is that within the local loop, the submodels only have access to a subset of the data fields after decomposition. For fields in `GRIDPOINT` representation this is usually a subset of vertical columns. Therefore, these entry points are only suitable for calculations, that do not require information about horizontally neighboring grid boxes. It is recommended that all submodels use `messy_physc` as the main entry point. Only in very few cases, it may be necessary to use `messy_convec` or `messy_vdiff` which occur earlier and may allow feedback to other processes that are calculated before `messy_physc`. Note that `messy_convec` is not called in the first time step (`lstart`).

2.2.3 The finalizing phase

For the finalizing phase, there is only one entry point (`messy_free_memory`). Its purpose is to deallocate submodel memory, which has been allocated in `messy_init_memory`. This is only required for memory that has been allocated directly by the submodel. The memory of objects defined by using the data transfer and export interface routines (currently stream elements, see section 2.3) and tracers is deallocated automatically and need not to be considered here.

A detailed summary of the implementation of all 4 MESSy layers can be found in the accompanying ECHAM5/MESSy reference card (`messy_echam5_ref.pdf`).

2.3 Memory management and data export

In the current version 0.9 of MESSy, the data transfer and export interface (including the memory management, see section 2.1.2) is based on the ECHAM5 *stream* implementation. Streams are sets of one- to four-dimensional Fortran95 arrays of type `REAL(DP)` used to store global data sets in various representations (see Table 2), and allow a flexible, `POINTER` based data handling. The contents of the streams can be easily output to netCDF files. Streams are very useful for sharing data between different submodels and between submodels and the base model (generic submodel main_data). However, the original ECHAM5 implementation has been significantly extended to meet the needs of a larger number of applications. The additional representations are indicated in Table 2.

When the model is executed in parallel mode, the data in stream elements of representation `GRIDPOINT`, `FOURIER`, `SPECTRAL`, and `LAGRANGIAN` are distributed automatically (i.e., decomposed) amongst the different processes. Stream elements of the representations `COLUMN`, `ARRAY1D`, and `SCALAR` exist for all parallel processes individually. Input / output and restart handling is only performed for the dedicated I/O process. Therefore synchronization of stream elements in these representations between the different processes running in parallel needs to be implemented depending on the application.

Another extension of the stream implementation allows the online calculation and output of the time average and (optionally) the corresponding standard deviation of stream elements, instead of instantaneous values. More detailed information is accessible via the MESSy web-pages.

2.4 Tracers

The generic submodel main_tracer is responsible for handling chemical compounds. Although a similar facility is included in the original ECHAM5, it has been completely re-implemented as generic submodel to achieve a higher flexibility and modularity. The handling of tracers in the model setup can be subdivided into two parts:

1. The setup of the overall framework for tracers, which is dependent on the base model, for instance, on the
   - domain (ocean or atmosphere, global or regional)
   - representation (spectral, gridpoint, Lagrangian)
   - dimension (box model, column model, 2D-, 3D-model)
   - resolution (size of the tracer data field)
   - decomposition in a parallel environment
2. The access of the different submodels to the tracers within this framework, e.g.,
   - definition of submodel-specific tracers
   - sharing tracers between different submodels

Within the overall framework, different sets of tracers (e.g., in different representations, like `GRIDPOINT`, or `LAGRANGIAN`), which coexist independently can be defined. Within such a set of tracers, a tracer consists of meta-data information describing the characteristics of the chemical compound and the field with the values (i.e., the tracer abundance or the amount of the tracer). Each tracer in a set is identified un-ambiguously by its unique name. The name of a tracer is composed of a base-name and an optional sub-name, in order to allow for the use of tracer classes, e.g., for 'tagged' tracers.

2.4.1 Overall Framework for tracers (tracer sets)

For setting up the overall framework, the SMCL of the generic submodel main_tracer provides the following subroutines (`messy_main_tracer.f90`):

- `new_tracer_set` initializes the framework for the meta-data information for a new tracer set. Up to 10 independent sets of tracers are supported.
- `setup_tracer_set` allocates the memory for the tracer data of one specific tracer set, after all tracers of the set have been defined.
- `get_tracer_set` gives access to meta-data and tracer data of one specific tracer set.
- `clean_tracer_set` removes one specific tracer set.
- `print_tracer_set` prints a summary of the meta-data information of one specific tracer set.
- `print_tracer_set_val` prints a summary of the tracer data of one specific tracer set.
- `tracer_error_str` returns an information string for the status flag returned by the subroutines of the SMCL of the generic submodel main_tracer.

In the current ECHAM5/MESSy implementation, these SMCL routines are called within the SMIL of the generic submodel main_tracer to set up two different, independent tracer sets: one in `GRIDPOINT` representation, and one in `LAGRANGIAN` representation. In particular, the `PUBLIC` SMIL routines (`messy_main_tracer_e5.f90`), which are called by the central submodel control interface (`messy_main_control_e5.f90`, see section 2.2), are:

Table 2: Different stream representations

REPRESENTATION	RANK	dimensions	automatic decomposition	output possible	original ECHAM5
GAUSSIAN = GRIDPOINT	4	latitude × level × species × longitude	yes	no	yes
GAUSSIAN = GRIDPOINT	3	latitude level × longitude	yes	yes	yes
GAUSSIAN = GRIDPOINT	2	latitude × longitude	yes	yes	yes
GAUSSIAN = GRIDPOINT	1	(unused)	yes	no	yes
FOURIER	4		yes	no	yes
SPECTRAL	3	level × Re/Im × wave number	yes	yes	yes
SPECTRAL	2	Re/Im × wave number	yes	yes	yes
LAGRANGIAN	1	number of air parcels	yes	yes	no
COLUMN	1	level	no	yes	no
ARRAY1D	1	arbitrary length	no	yes	no
SCALAR	0	scalar	no	yes	no

- `main_tracer_new_tracer` initializes the framework for the meta-data information of the two tracer sets.
- `main_tracer_init_memory` initializes the memory for the tracer data of the two tracer sets, after all submodels have contributed their individual tracer definitions.
- `main_tracer_free_memory` deallocates the memory used for the tracer data of the two tracer sets.
- `main_tracer_init_tracer` organizes the tracer initialization (see section 2.4.3) of the two tracer sets.

For output and decomposition of tracers among the different processes in a parallel environment, the generic submodel main_tracer is linked to the data transfer/export interface.

2.4.2 Tracer access of submodels

For the access to tracers within the specific submodels, the SMCL of the generic submodel main_tracer (`messy_main_tracer.f90`) provides the following subroutines:

- `new_tracer` is used to define a new tracer as member of a specific tracer set.
- `get_tracer` is used to access meta-data information and tracer data of a tracer in a specific tracer set.

In addition, the SMIL of the generic submodel main_tracer (`messy_main_tracer_e5.f90`) provides the following subroutines:

- `tracer_init` is used to initialize a (sub-)set of tracers via the data import interface.
- `tracer_halt` is used to output a status information and to terminate the model, if a subroutine of the generic submodel main_tracer returned an error status.

Tracer access within a specific submodel is exclusively performed within the SMIL of the submodel.

2.4.3 Tracer initialization

The initialization of tracers in ECHAM5/MESSy is a three stage process, which is individually controllable for each tracer (in the following, given options in parentheses apply to the subroutine call of `new_tracer(...)`):

1. The tracer is initialized from a restart file, if all of the following conditions are met:
   - ECHAM5/MESSy is in the first time step after starting in rerun mode (`lrerun=.TRUE.`), e.g., in job chains.
   - The tracer has been dumped to the restart file, i.e., the data is available (`..., lrestart=.true.,...`). If the tracer is not available in the restart file, but the tracer is labeled to be absolutely required (`..., lcontnorest=.false.,...`) the model is terminated.
2. The tracer is initialized via the data import interface used in the SMIL (`<submodel>_init_tracer`) of the specific submodel (`CALL tracer_init(modstr)`), if all of the following conditions are met:
   - ECHAM5/MESSy is in the first time step (very first time step or first time step in rerun mode, i.e., `lstart=.TRUE.` or `lrerun=.TRUE.`).
   - The tracer is named in one of the `&REGRID` namelists in `<submodel>_t.nml`.
   - The tracer has not already been initialized from a restart file, or the tracer has been already initialized from a restart file, but the tracer is labeled to ignore this (`..., lforce_init=.true.,...`).
   - The tracer is not manually labeled to be already initialized (`..., linit=.true.,...`).

   Note that with this construction the first submodel with an entry for a specific tracer in the corresponding `<submodel>_t.nml` determines the initialization of this tracer.

3. The tracer is set to a constant value (..., vini=<value>,...) with the default being 0, if it is not yet initialized.

2.5 Available submodels

A comprehensive and up to date list of available MESSy submodels with further detailed information and contact persons is available at the MESSy web-pages (http://www.messy-interface.org).

2.6 Directory structure

The directory structure of the ECHAM5/MESSy distribution is that of the ECHAM5 distribution with an additional subdirectory messy, according to the MESSy standard. The SMIL directory is messy/e5.

2.7 Input files

Input files for ECHAM5/MESSy can be divided into 2 groups: Files for the base model ECHAM5 and files for the MESSy submodels. In the run-script xmessy, the shell variables INPUTDIR_ECHAM5_INI and INPUTDIR_MESSY are used to specify the two input base directories, if the default directory structure does not apply. This can be useful if the ECHAM5 input is already available in a read-only directory and the latest MESSy submodel data must be saved elsewhere. In the run-script xmessy the default directories are set according to the currently supported hosts. Running **xmessy -h** gives more information on supported hosts.

Within the INPUTDIR_ECHAM5_INI directory, the ECHAM5 specific input files need to be grouped into subdirectories according to their horizontal resolution, e.g., T21, T31, T42, T63, T85, T106, etc.

The subdirectory tree of INPUTDIR_MESSY is organized as

```
raw/<submodel>/*.nc
raw/<submodel>/misc/*
raw/<submodel>/README
```

i.e., each MESSy submodel has one subdirectory under raw with the subdirectory name being exactly identical to the name of the submodel. In the subdirectory tree raw, gridded data is stored in their original resolution as netCDF files (*.nc). Those input data which are not available as netCDF files (e.g., because they are not defined on a global grid), are stored in the submodel subdirectory <submodel>/misc/. In the file README in the submodel subdirectory the submodel specific input data are briefly explained and references are given.

Since gridded MESSy data are usually imported via the data import interface, i.e., rediscretized to the actual model resolution, data in its original resolution is sufficient.

2.8 MESSy utilities

The ECHAM5/MESSy distribution contains several utilities to facilitate several aspects of the model development, of the model application, and of the post processing and the data visualization. These utilities can be classified into two categories: one group for direct user interaction, and one group only used indirectly.

2.8.1 User utilities

2.8.1.1 xecham is a tcsh-script located in the base directory of the ECHAM5/MESSy distribution (from where it must also be started). This script guides the user automatically through all steps of installing and running ECHAM5/MESSy. So far, it has been used on PC/Linux and Compaq-Alpha/OSF1 platforms.

2.8.1.2 mchlog is a ksh-script located in messy/util for finding the correctly labeled changes in the code (see section 5.6).

2.8.1.3 nc2mc is a tcsh-script in messy/util for creating a multi-netCDF descriptor file for Ferret (http://ferret.wrc.noaa.gov/Ferret/). With such a meta-file, a time-series of data which is subdivided into several netCDF files (like the ECHAM5/MESSy output), can be addressed as one comprehensive dataset in Ferret. More information can be obtained with nc2mc -h.

2.8.1.4 ncdx is a Fortran95 program located in messy/box to make netCDF files (such as the ECHAM5/MESSy output) OpenDX (http://www.opendx.org) compliant.

2.8.1.5 nc2dxmc is a tcsh-script in messy/util for creating a multi-netCDF descriptor file for OpenDX (http://www.opendx.org/). With such a meta-file, a time-series of data which is subdivided into several netCDF files (like the ECHAM5/MESSy output), can be addressed as one comprehensive dataset in OpenDX by using the macro mc4dx. More information can be obtained with nc2dxmc -h.

2.8.1.6 ncregrid is the box-model of the MESSy data import interface (and therefore located in messy/box), i.e., the stand-alone version for offline rediscretization of 2D and 3D data between arbitrary, rectangular global, geo-hybrid grids.

2.8.2 Auxiliary utilities

The auxiliary utilities, which are not for direct users interaction, are all located in messy/util:

2.8.2.1 efchk, lst2log.gawk, efchk-sum are scripts used for the comprehensive forcheck (http://www.forcheck.nl) analysis of the code (gmake check).

2.8.2.2 messy_check_... are tcsh-scripts to perform some (only the most important) tests for MESSy conformity of the code (`gmake messycheck`).

2.8.2.3 zip.sh, zip1r.sh, zipall.sh are sh-scripts for packing the distribution (`gmake zip`, `gmake zip1r`, `gmake zipall`).

2.8.2.4 sfmakedepend.pl is a PERL-script for generating the file dependencies for (g)make (see `http://people.arsc.edu/~kate/Perl/sfmakedepend`, this original file has been modified, however).

3 Installation of ECHAM5/MESSy

Before the installation of the ECHAM5/MESSy distribution can be successful, the following software packages must be available / installed on the computer system:

- a C-compiler
- a Fortan95 compiler
- the netCDF library version 3.5.1b or higher (`http://www.unidata.ucar.edu/packages/netcdf/`)
- if ECHAM5/MESSy should be run in parallel mode (which is highly recommended), the Message Passing Interface (MPI) library must be available (`http://www-unix.mcs.anl.gov/mpi/index.html`, `http://www-unix.mcs.anl.gov/mpi/mpich/`)
- ECHAM5/MESSy can optionally make use of pre-installed libraries `BLAS` and `LAPACK`. If these libraries are not available, they are automatically build within the ECHAM5/MESSy distribution (subdirectories `blas` and `lapack`).
- If one of the MESSy submodels MECCA (Module Efficiently Calculating the Chemistry of the Atmosphere, Sander et al. [2005]) or SCAV should be used, the Kinetic Pre-Processor (KPP, Damian et al. [2002], `http://www.cs.vt.edu/~asandu/Software/KPP`) must be available.
- Installation of the post-processing / visualization software Ferret (`http://ferret.wrc.noaa.gov/Ferret/`) and / or OpenDX (`http://www.opendx.org`) is recommended, although not required for using ECHAM5/MESSy.

If these prerequisites are met, the installation of the ECHAM5/MESSy distribution on supported platforms is straightforward:

1. Unpack the ECHAM5/MESSy distribution with
   `unzip echamX.X.XXx_messy_Y.Yy.zip`
   and change into the newly created directory (`X.X.XXx` denotes the version of the used ECHAM5 base model and `Y.Yy` the MESSy version, e.g., echam5.2.02a_messy_0.9.zip).
2. Edit the file
   `config/mh-<arch>`
   for your system (`<arch>`) and and adapt the required system specific variables (compiler, compiler flags, etc.) and the required host specific settings (paths to netCDF- and MPI-libraries and include-files). Host specific settings for the same system can be structured using a case-construct:

```
case ‘uname -n‘ in
  <host_1>)
      # settings for host ’host_1’
      ;;
  <host_2>)
      # settings for host ’host_2’
      ;;
  *)
      # default settings
      ;;
esac
```

3. Update the Fortran95 code for those submodels for which the code is (partly) generated automatically, according to the users specifications. More details on automatic code generation can be found in the respective descriptions of the submodels.
4. Configure the ECHAM5/MESSy distribution for the used system with
   `./configure`
   Optionally, the following options can be used:
   - `--host=...` for using a cross-compiler (i.e., the platform where ECHAM5/MESSy is run is different from the platform it is compiled on)
   - `--disable-MESSY` to configure ECHAM5 without MESSy
   - `--disable-MPI` to configure ECHAM5/MESSy without MPI, i.e., for a single process mode
5. Build the executable with:
   `gmake`
   or
   `make`
   dependent on your system. Note that
   `(g)make help`
   provides you a list of further build-targets.

The currently supported and tested platform / compiler combinations are listed in Table 3. A more up to date list is available on the MESSy web-pages.

If ECHAM5/MESSy should be ported to a new platform, in principle only a system specific file `config/mh-<arch>` for the new platform must be created.

Table 3: Supported platforms for ECHAM5/MESSy.

Hardware	System	Fortran95 Compiler	remarks
PC	Linux 2.4	Lahey/Fujitsu 6.2a	
PC-Cluster	Linux 2.4	Lahey/Fujitsu 6.2a	
PC(-Cluster)	Linux 2.4	Intel ifort 8.1	unresolved compiler issues; compiler crashes
Compaq/Alpha	OSF1 5.1a	native f95	unresolved issues with runtime checks; code runs
IBM-p690	AIX	mpxlf95_r	
NEC-SX6	SUPER-UX	rev276	cross compiler on Linux 2.4
MAC	Darwin	IBM	cooperatively supported

4 Running ECHAM5/MESSy

Before ECHAM5/MESSy can be run successfully, the initialization data for ECHAM5 in the desired grid resolution and the MESSy input data must be available (see section 2.7).

The central instance of the user interface (see section 2.1.5) for running ECHAM5/MESSy is the run-script (sh-script) `xmessy`, which automatically detects

- the operating system:
  - OSF1
  - Linux
  - AIX
  - SUPER-UX
  - Darwin
- the job scheduling system, if `xmessy` has been submitted to such a system:

  LL (IBM Load Leveler)

  NQS I (Network Queuing System I)

  NQS II (Network Queuing System II)

  SGE (Sun Grid Engine, formerly CODINE)

  SCORE
- the hostname

For the known combinations of hostname, operating system, and job scheduling system, `xmessy` can be applied immediately. To setup a model experiment, the user has to

- set the embedded flags in the header section for the queuing system `xmessy` should be submitted to. These settings have no meaning if `xmessy` is run in the foreground or background.
- set the number of processes for the parallel environment (`NCPUS`). This setting is ignored, if the number of processes is requested as a resource via the queuing system. In case ECHAM5/MESSy has been compiled without MPI (i.e., configured with the option `--disable-MPI`), `NCPUS` must be set to zero.
- set some basic settings for the base model, e.g.,
  - the vector blocking and decomposition (`NPROMA`, `NPROCA`, `NPROCB`),
  - the horizontal resolution (`HRES`),
  - the vertical resolution (`VRES`),
  - the start-date of the integration (`START_YEAR`, `START_MONTH`, `START_DAY`)
  - the stop-date of the integration, if activated in the namelist-file chosen by `NML_ECHAM` (`STOP_YEAR`, `STOP_MONTH`, `STOP_DAY`)
  - the mode of ECHAM5 (e.g., column mode, nudging mode, which sea surface temperature, etc.).
- set the base directory of the ECHAM5/MESSy distribution, the data input path for ECHAM5 and the MESSy submodels, the working directory (where intermediate and output files are written to), the directory containing the namelist files of the submodel, and the name of the executable. These settings have only to be specified, if the default settings are not applicable.
- select the submodels which should be activated.
- choose the namelist files for the base model and the submodels. For control of the activated submodels, the user has to edit the selected namelist files in the namelist file directory. All variables in the namelist files (beginning with $) will be replaced by the respective shell-variables defined in `xmessy`.

After these settings, `xmessy` can be run in the foreground (output to the current shell), background (redirection to a log-file is possible according to the syntax of the used shell), or be submitted to a job scheduling system (output redirection via embedded flags of the used queuing system). Up-to-date information on hostnames, operating systems, and queuing systems can be obtained by running `xmessy -h` in foreground.

`xmessy` terminates with an error message, if, for instance,

- the continuation of a job chain has been triggered, however, the required rerun files are not present,
- a core file has been found, i.e., the executable crashed during execution,
- `xmessy` has been started in foreground or background on a platform where submission to a job scheduling system is required,

- the number of processes cannot be determined, because the user did not request it by using the embedded flags for the queuing system.

On systems not yet known to `xmessy`, the run-script will terminate if

- the operating system is not recognized or not yet supported,
- the hostname is not recognized or not yet supported,
- the parallel environment is not recognized or not yet supported.

Since these information are required at several places in the script, the corresponding error messages are labeled with a number, in order to be able to find the place where the error occurred in the script. This is especially helpful for the implementation of new operating systems and/or hostnames, which is best done iteratively.

5 ECHAM5/MESSy coding guidelines

The quality assurance (QA) of new ECHAM5/MESSy code is applied on three levels,

1. the ISO/IEC-1539-1 Fortran95 standard conform implementation of MESSy which is analyzed using `forcheck` (`http://www.forcheck.nl`)
2. the MESSy standard conform implementation of all submodels
3. some coding guidelines as listed in the following

5.1 Contribution of new submodels

In order to cope with the increasing complexity, and to meet the overall requirements of MESSy, future contributions of new submodels must comprise

- a self-consistent box model
- a complete list of input and output parameters
- a brief manual (what does it do?) and web page

5.2 File names and directories

- The file names must be `messy_<submodel>.f90` for the submodel core file and `messy_<submodel>_e5.f90` for the submodel interface file. If the submodel needs additional files, they can be called `messy_<submodel>_*.f90` and `messy_<submodel>_*_e5.f90`, respectively.
- Submodel interface files (SMIL, `*_e5.f90`) must be in the `messy/e5/` directory and core (SMCL) files in the `messy/src/` directory.
- Memory files (`*_mem.f90` or `*_mem_e5.f90`) must not contain any SUBROUTINEs or FUNCTIONs. They must be used at least twice but not only by SMCL and corresponding SMIL file.

5.3 Data exchange via Fortran95 USE statements

- Core files (`messy/src/*.f90`) must not USE modules from interface files (`messy/e5/*_e5.f90`) or base model files (`modules/mo_*.f90`).
- Core files (`messy/src/*.f90`) must not USE modules from other submodels.
- Data objects (stream elements) from other submodels must be selected via the `CPL`-namelist.

5.4 Data exchange via the data transfer and output interface and tracers

- The memory stream must have the name of the submodel. If more than one stream is created, additional streams can be called `<submodel>_*`.
- Submodels are allowed to use streams created by other submodels (currently with `get_stream`). After checking that the stream was successfully opened, they can read data from the stream elements in that stream with `get_stream_element`. However, they must neither add stream elements to streams of other submodels, nor must they change existing stream elements of other submodels.
- Submodels are allowed to change the tendencies of any tracer, including those that were created by other submodels.
- `get_tracer`, `get_stream`, or `get_stream_element` must not be used inside the time loop. The respective pointers must be set during the coupling-initialization.

5.5 Parallel environment

- STOP statements in core (SMCL) files are not allowed. As alternative, a non-zero INTEGER error status must be returned to the calling SMIL routine, in order to terminate the system in the parallel environment from there.

5.6 Labels

All lines of MESSy-code in the base model and all lines in MESSy files altered by anybody else than the submodel maintainer must be clearly labeled. The label

```
! ii_aa_yyyymmdd
```

consists of the prefix of the institution (`ii`), followed by an underscore, the author's initials (`aa`), a second underscore, and the date of change in the 8-digit format (year, month, day), e.g., `mz_pj_20040727`. A list of institutions and authors can be found in the file `AUTHORS_MESSY` of the ECHAM5/MESSy-distribution. If just one line is changed in the code, the comment can be added at the end of the line. If a whole block of Fortran code is added or changed, it should be enclosed between two such comment lines, the first ending with `+`, the second ending with `-`, e.g.,

```
! mz_pj_20040727+
<changed code>
! mz_pj_20040727-
```

Within the block, the reasons for changing the code must be explained.

With the tool `./messy/util/mchlog` (see section 2.8.1) correctly commented lines and blocks in the source code can be traced.

5.7 Loops

To make loops easily recognizable, they should be marked with a Fortran label. Loop variables must start with the letter `j`. Depending on the type of loop (e.g., level, tracer, or vector), the names `jk`, `jt`, or `jp` should be used, respectively:

```
level_loop: DO jk=1,nlev
  tracer_loop: DO jt=1,ntrac_gp
    vector_loop: DO jp=1,kproma
      ...
    END DO vector_loop
  END DO tracer_loop
END DO level_loop
```

If global fields in `GRIDPOINT` representation need to be accessed, the following variable names are recommended:
`jxg = ilon(jp,jrow)` x-value, global (longitude)
`jyg = ilat(jp,jrow)` y-value, global (latitude)
The index-fields `ilon` and `ilat` can be accessed via

```
USE mo_geoloc, ONLY: ilon, ilat
```

The current row within the local loop is called `jrow` or `nrow(2)`. Its value lies between 1 and `ngpblks`.

```
USE mo_control, ONLY: nrow
...
INTEGER :: jrow
...
jrow = nrow(2)
...
```

The value of the vector loop length (`kproma`), which is in general dependent on the row within the local loop, can be calculated with this code:

```
USE mo_decomposition, ONLY: &
  dcl => local_decomposition
...
IF ( jrow == dcl%ngpblks ) THEN
  kproma = dcl%npromz
ELSE
  kproma = dcl%nproma
ENDIF
```

Finally, the number of tracers can be accessed with

```
USE messy_main_tracer_mem_e5, ONLY: &
  ntrac_gp, ntrac_lg
```

for tracers in `GRIDPOINT` (`_gp`) and `LAGRANGIAN` (`_lg`) representation, respectively.

6 Outlook

The following details are subject to change for future versions of ECHAM5/MESSy:

- The ECHAM5 stream implementation (which is currently the base of the MESSy data transfer and export interface) will be replaced by a much more flexible and self-consistent (i.e., ECHAM5 independent) *memory channel object* implementation:
  - coded as generic MESSy submodel for higher modularity
  - only one central namelist file controlling all aspects of data output for all memory channels and memory channel objects (including tracers): output frequency, output files, output format, output format specific conventions (e.g., OpenDX compliant output without ncdx), etc.
  - tracers are consistently linked, i.e., input/output parameters (`lrestart`, `lcontnorest`) in tracer definitions (see section 2.4.3) will become obsolete
- User interface (see section 2.1.5): All namelist files are named `<submodel>.nml` and `<submodel>_t.nml`. Different sets for different model setups will be grouped into various subdirectories of `messy/nml`. Only the namelist file subdirectory has to be selected in the run-script `xmessy`. The namelist file subdirectories will also include netCDF-file specific regridding namelist files.
- The compilation utility `sfmakedepend.pl` will recognize dependencies of the SMIL modules to SMCL Fortran95 module files.

- The BMIL of ECHAM5/MESSy will be reorganized such that the SMIL files do not require further direct connection (i.e., USE of) of ECHAM5 parameters and routines (e.g., see section 5.7). The link is performed via `messy_main_data_e5` and a few additional files of the BMIL.
- The subroutines `<submodel>_read_nml` and `<submodel>_read_nml_e5` will be renamed to `<submodel>_read_nml_ctrl` and `<submodel>_read_nml_cpl`, respectively.
- A pre-regridding facility for MESSy input data will be included in the distribution. It can be activated, via an additional switch in the run-script `xmessy`. This will automatically select the respective input-path and adjust the required regridding-namelists. The input-path with (horizontally) pre-regridded netCDF files will be named as the horizontal resolution (`T21`, `T42`, `T63`, ...). It will be parallel to the `raw` directory and organized in the same way (see section 2.7). Pre-regridding will speed up the initialization phase of the model when using large datasets, and also the integration phase when time dependent offline data (with high frequency) are used as boundary conditions.
- The output of control runs for various standard model setups will be made available in a web-accessible database.

Acknowledgments

For helpful suggestions about this manual we would like to thank Joachim Buchholz, Astrid Kerkweg, Holger Tost, and Michael Traub.

References

Damian, V., Sandu, A., Damian, M., Potra, F., and Carmichael, G. R.: The kinetic preprocessor KPP - a software environment for solving chemical kinetics, Comput. Chem. Eng., 26, 1567–1579, 2002.

Jöckel, P., Sander, R., Kerkweg, A., Tost, H., and Lelieveld, J.: Technical Note: The Modular Earth Submodel System (MESSy) - a new approach towards Earth System Modeling, Atmos. Chem. Phys., this issue, 2005.

Sander, R., Kerkweg, A., Jöckel, P., and Lelieveld, J.: Technical Note: The new comprehensive atmospheric chemistry module MECCA, Atmos. Chem. Phys., this issue, 2005.