MESSy TRACER User Manual

Patrick Jöckel, Astrid Kerkweg,
Joachim Buchholz, Holger Tost, Rolf Sander &
Andrea Pozzer

for the MESSy TRACER submodel

Air Chemistry Department
Max-Planck Institute of Chemistry
PO Box 3060, 55020 Mainz, Germany
`joeckel@mpch-mainz.mpg.de`

This manual is part of the electronic supplement of our article "Technical note: Coupling of chemical processes with the Modular Earth Submodel System (MESSy) submodel TRACER" in Atmos. Chem. Phys. (2008), available at: `http://www.atmos-chem-phys.net`

Date: February 12, 2008

Contents

1 Introduction 3

2 The meta-information structure 3

3 The file `messy_main_tracer.f90` 4

3.1 Tracer set routines to be called from the BMIL 4

3.1.1 The subroutine `new_tracer_set` 4

3.1.2 The subroutine `copy_tracer_set` 5

3.1.3 The subroutine `setup_tracer_set` 5

3.1.4 The subroutine `get_tracer_set` 5

3.1.5 The subroutine `clean_tracer_set` 6

3.1.6 The subroutine `print_tracer_set` 6

3.1.7 The subroutine `print_tracer_set_val` 6

3.1.8 The subroutine `main_tracer_read_nml_ctrl` 7

3.2 Tracer routines to be called from the SMIL of MESSy submodels 7

3.2.1 The subroutine `new_tracer` 7

3.2.2 The subroutine `set_tracer` 8

3.2.3 The subroutine `get_tracer` 8

3.2.4 The subroutine `get_tracer_list` 9

3.2.5 The subroutine `tracer_iniflag` 10

3.2.6 The function `tracer_error_str` 10

3.2.7 The function `param2string` 10

3.2.8 The subroutine `full2base_sub` 11

4 A simple example application 11

5 The file `messy_main_tracer_family.f90` 13

5.1 The subroutine `tracer_family_read_nml_ctrl` 13

5.2 The subroutine `tracfamily_init` 13

5.3 The subroutine `tracfamily_newtrac` 13

5.4 The subroutine `tracfamily_initmode` 13

5.5 The subroutine `tracfamily_meta` 13

5.6 The subroutine `tracfamily_1_f2t` 14

5.7 The subroutine `tracfamily_1_t2f` 14

5.8 The subroutine `tracfamily_2_rsc` 14

5.9 The subroutine `tracfamily_2_sum` 15

5.10 The subroutine `tracfamily_freemem` 15

6 The file `messy_main_tracer_pdef.f90` 15

6.1 The subroutine `tracer_pdef_read_nml_ctrl` 15

6.2 The subroutine `tracpdef_initmem` 15

6.3 The subroutine `tracpdef_settings` 15

6.4 The subroutine `tracpdef_airmass` 15

6.5 The subroutine `tracpdef_integrate` 16

6.6 The subroutine `tracpdef_freemem` 16

6.7 The subroutine `tracpdef_print` 16

1 Introduction

This documentation describes some more details of the MESSy submodel TRACER for the chemical coupling of processes in Earth System Models. Sect. 2 explains the tracer *meta-information* structure and how to expand it. A reference for the interface routines of the submodel follows. In most cases, the routines described in Sect. 3 are sufficient to apply TRACER. An example is illustrated in Sect. 4. The submodel core layer (SMCL) routines of the TRACER submodels TRACER_FAMILY and TRACER_PDEF are described in Sect. 5 and Sect. 6, respectively.

In the following description of the Fortran95 code, two parameters from the module `messy_main_constants_mem.f90` are used, one for the precision of real numbers:

```
INTEGER, PARAMETER :: DP = SELECTED_REAL_KIND(12,307)
```

and one for the length of strings:

```
INTEGER, PARAMETER :: STRLEN_MEDIUM = 24
```

The documented code is also part of the MESSy distribution version 1.5.

2 The meta-information structure

The *meta-information* of the tracers in a tracer set are stored in a concatenated list of Fortran95 structures:

```
TYPE t_trinfo_list
   TYPE(t_trinfo)               :: info
   TYPE(t_trinfo_list), POINTER :: next
END TYPE t_trinfo_list
```

The *meta-information* of one tracer is split into two parts:

```
TYPE t_trinfo
   TYPE(t_ident) :: ident       ! IDENTIFICATION
   TYPE(t_meta)  :: meta        ! ADDITIONAL META-INFORMATION
END TYPE t_trinfo
```

The first part (`t_ident`) is for the identification of the tracer and the second part (`t_meta`) to store additional *meta-information.* The tracer identification contains:

- a unique *fullname* consisting of a *basename* and an additional optional *subname*,
- the name of the submodel which has defined the tracer in the tracer set,
- a unique index, which is the number of the tracer in the tracer set that can also be used to address the corresponding *data*,
- the *medium* of the tracer; the following integer parameters are pre-defined: AIR=1, AEROSOL=2, CLOUD=3, OCEAN=4, LAKE=5, RIVER=6, LANDICE=7, SEAICE=8, VEGETATION=9,
- the *quantity* describing the abundance of the tracer; the following integer parameters are pre-defined: AMOUNTFRACTION=1, NUMBERDENSITY=2, CONCENTRATION=3,
- the *unit* of the tracer *data*,
- the *type* of the tracer; the following integer parameters are pre-defined: SINGLE=0, FAMILY=1, ISOTOPE=2.

The additional *meta-information* contains one (logical) flag to store the initialisation state of the tracer *data* and three *meta-information* containers (Fortran95 arrays of rank one), for integer, real and string information, respectively. The meaning of a container content is solely defined by its position in the respective array (i.e., the container number); currently the following container numbers are pre-defined:

container number	value	meaning	possible container content
`I_ADVECT`	1	switch for advection	**ON**, OFF
`I_CONVECT`	2	switch for convection	**ON**, OFF
`I_VDIFF`	3	switch for vertical diffusion	**ON**, OFF
`I_WETDEP`	4	switch for wet deposition	ON, **OFF**
`I_DRYDEP`	5	switch for dry deposition	ON, **OFF**
`I_SEDI`	6	switch for sedimentation	ON, **OFF**
`I_SCAV`	7	switch for scavenging	ON, **OFF**
`I_MIX`	8	switch for turbulent mixing	**ON**, OFF
`I_FORCE_COL`	9	switch for forcing in column mode	ON, **OFF**
`I_INTEGRATE`	10	switch for time integration	**ON**, OFF
`I_TIMEFILTER`	11	switch for time filter	**ON**, OFF
`I_FORCE_INIT`	12	switch for re-initialisation after restart	ON, **OFF**
`I_AEROSOL_METHOD`	13	method of aerosol dynamical model	**MODAL**, BIN
`I_AEROSOL_MODE`	14	number of aerosol mode or bin	(0)
`I_AEROSOL_SOL`	15	aerosol solubility flag	**ON**, OFF
`R_MOLARMASS`	1	molar mass of species	(0.0) g/mol
`R_HENRY`	2	effective henry's law coefficient	(0.0) mol/L/atm
`R_DRYREAC_SF`	3	coefficient for dry reaction with surface	e.g., **0.0**, 0.1, 1.0
`R_VINI`	4	initial value for tracer *data*	(0.0)
`R_AEROSOL_DENSITY`	5	aerosol component density	(0.0)
`S_AEROSOL_MODEL`	1	name of associated aerosol dynamical model	

Text in bold-face or in parentheses shows the default values; OFF=0, ON=1, MODAL=2, and BIN=3 are pre-defined integer parameters. The container number names begin with "I_", "R_" and "S_" for integer, real and string containers, respectively.

To add new containers, the following steps are required (`messy_main_tracer.f90`, X is either I, R or S):

- add new container number parameter with MAX_CASK_X + 1
- increase MAX_CASK_X by one
- add descriptive string to NAMES_CASK_X
- add default container content to DEFAULT_CASK_X

3 The file `messy_main_tracer.f90`

The subroutines and functions in this file constitute the main interface routines for the application of TRACER from within a model. They are divided into two groups: the first group (to be called from the basemodel interface layer (BMIL)) to provide the overall framework for *tracer sets*, and the second group (to be called from the submodel interface layer (SMIL)) of MESSy submodels to handle individual *tracers*.

In the following description, the Fortran95 variable `status` defines an INTENT(OUT) variable of type INTEGER, which returns the status information of the respective subroutine. The `status` is 0, if the routine was successful, and > 0, if an error occurred. The value of `status` can be transformed by the function `tracer_error_str` into an error message.

3.1 Tracer set routines to be called from the BMIL

3.1.1 The subroutine `new_tracer_set`

The subroutine `new_tracer_set` defines a new tracer set and generates the *meta-information* framework for the set.

`SUBROUTINE new_tracer_set`		`(status ,setname ,l_enable)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
setname	CHARACTER(LEN=*)	IN	name of the tracer set
l_enable	LOGICAL	IN	enable or disable tracer set

The name of a tracer set must be unique.

With the switch `l_enable` a tracer set can be enabled (.TRUE.) or disabled (.FALSE.) during the *initialisation phase* of the model simulation. Depending on the model setup, not all available (=defined) tracer sets might be always enabled. The routines accessing tracers (Sect. 3.2) of disabled tracer sets will always return `status=0`.

3.1.2 The subroutine `copy_tracer_set`

The subroutine `copy_tracer_set` copies the *meta-information* of a complete tracer set (including the *meta-information* of its tracers) into a new tracer set.

`SUBROUTINE copy_tracer_set`		`(status ,oldset ,newset)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
oldset	CHARACTER(LEN=*)	IN	name of the tracer set to be copied
newset	CHARACTER(LEN=*)	IN	name of the new tracer set

This can be used to specify tracer sets, which are identical w.r.t. their tracer *meta-information*, however, different w.r.t. their *representation*, e.g., tracer sets with different grid structures.

3.1.3 The subroutine `setup_tracer_set`

The subroutine `setup_tracer_set` allocates memory for a tracer set.

`SUBROUTINE setup_tracer_set`		`(status ,setname ,dim ,nt ,l_tfstd ,l_init)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
setname	CHARACTER(LEN=*)	IN	name of the tracer set
dim	INTEGER, DIMENSION(3)	IN	*representation* dimension lengths
nt	INTEGER	IN	number of *data instances*
l_tfstd	LOGICAL	IN	.TRUE. for standard *instances*
l_init	LOGICAL	IN	initialisation protector

Up to three dimensions (`dim`) can be used to associate a *representation* (e.g., a 3-dimensional spatial grid) to a tracer set. If the *representation* is less than 3-dimensional, the remaining dimension lengths must be set to 1.

The number of *data instances* is usually used for different stages of the time integration scheme (e.g., leap-frog with filter). In case `nt` is 3 or larger, the switch `l_tfstd` (=.TRUE.) associates a special meaning to the first three instances, namely the tracer *data* at time step t, the tracer tendency, and the tracer *data* at time step $t - \Delta t$, respectively (Δt is the model time step length).

The switch `l_init` (=.FALSE.) can be used to protect the tracer *data* of all tracers in this set from potential tracer initialisation procedures during the *initialisation phase* of the model (in the BMIL).

3.1.4 The subroutine `get_tracer_set`

The subroutine `get_tracer_set` sets references to tracer sets.

`SUBROUTINE get_tracer_set`		`(status [,setid] [,setname] [,trlist] [,ti]`
		`[,ntrac] [,xt] [,xtte] [,xtm1] [,xmem]`
		`[,l_tfstd] [,l_init] [,l_enable])`	

name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
optional arguments:			
setid$^{*)}$	INTEGER	IN	id of tracer set
setname$^{*)}$	CHARACTER(LEN=*)	IN$^{*)}$	name of the tracer set
trlist	TYPE(t_trinfo_list)	POINTER	*meta-information* list
ti	TYPE(t_trinfo_tp), DIMENSION(:)	POINTER	*meta-information* array
ntrac	INTEGER	OUT	number of tracers in set
xt	REAL(DP), DIMENSION(:,:,:,:,:)	POINTER	1st *instance*
xtte	REAL(DP), DIMENSION(:,:,:,:,:)	POINTER	2nd *instance*
xtm1	REAL(DP), DIMENSION(:,:,:,:,:)	POINTER	3rd *instance*
xmem	REAL(DP), DIMENSION(:,:,:,:,:)	POINTER	> 3 or all *instances*
l_tfstd	LOGICAL	OUT	standard *instances* ?
l_init	LOGICAL	OUT	initialisation protector ?
l_enable	LOGICAL	OUT	enabled or disabled ?

$^{*)}$Note: If `setid` is present, `setname` is optional and INTENT(OUT). Otherwise, `setname` is mandatory and INTENT(IN).

`trlist` returns the tracer *meta-information* as a concatenated list of Fortran95 pointer structures, whereas `ti` returns the same information as 1-dimensional array with the *tracer index* as array index.

The total number of tracers defined in the set by all submodels (with the routine `new_tracer`) is `ntrac`.

`xt`, `xtte`, `xtm1` and `xmem` are pointers to the *data* memory. `xt` always points to the first data *instance*. In case `l_tfstd` is not set (=.FALSE.), or the number of *instances* is less than three, `xtte` and `xtm1` are nullified pointers, and `xmem` points to all *instances* > 1. In case `l_tfstd` is .TRUE. and the number of *instances* is three or larger, `xt`, `xtte` and `xtm1` point to *instances* 1 to 3 respectively, and `xmem` to all *instances* > 3 (if available; otherwise the pointer is nullified).

The initialisation protector and the activity status (enabled or disabled) of the tracer set can be retrieved with `l_init` and `l_enable`, respectively.

3.1.5 The subroutine `clean_tracer_set`

The subroutine `clean_tracer_set` removes a tracer set from memory.

`SUBROUTINE clean_tracer_set`		`(status ,setname)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
setname	CHARACTER(LEN=*)	IN	name of the tracer set

The *data* memory of the tracer set is deallocated and the *meta-information* of all tracers in the tracer set are deleted.

3.1.6 The subroutine `print_tracer_set`

The subroutine `print_tracer_set` prints a summary of all tracer sets (tracer *meta-information* only).

`SUBROUTINE print_tracer_set`	`()`

This routine can be used to output the tracer meta-information of all tracer sets after the initialisation or for debugging.

3.1.7 The subroutine `print_tracer_set_val`

The subroutine `print_tracer_set_val` prints the range of tracer *data* values (all *instances*) of all tracers in all tracer sets.

SUBROUTINE `print_tracer_set_val`	`()`

This routine can be used to output the tracer information of all tracer sets after the tracer sets have been set up, or for debugging.

3.1.8 The subroutine `main_tracer_read_nml_ctrl`

The subroutine `main_tracer_read_nml_ctrl` reads the tracer CTRL namelist, checks it, and initialises global variables.

SUBROUTINE `main_tracer_read_nml_ctrl`		`(status ,iou)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
iou	INTEGER	IN	Fortran95 input unit

3.2 Tracer routines to be called from the SMIL of MESSy submodels

3.2.1 The subroutine `new_tracer`

The subroutine `new_tracer` defines a new tracer in a set and optionally sets the tracer *meta-information*.

SUBROUTINE `new_tracer`		`(status ,setname ,basename ,submodel [,idx] [,subname] [,longname] [,unit] [,medium] [,quantity] [,type] [,cask_i] [,cask_r], [,cask_s])`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
setname	CHARACTER(LEN=*)	IN	name of the tracer set
basename	CHARACTER(LEN=*)	IN	basename of the tracer
submodel	CHARACTER(LEN=*)	IN	name of submodel which defines the tracer
optional arguments:			
idx	INTEGER	OUT	index of tracer in tracer set
subname	CHARACTER(LEN=*)	IN	subname of the tracer
longname	CHARACTER(LEN=*)	IN	string for more information
unit	CHARACTER(LEN=*)	IN	unit of the tracer *data*
medium	INTEGER	IN	medium of the tracer
quantity	INTEGER	IN	quantity of the tracer
type	INTEGER	IN	type of the tracer
cask_i	INTEGER, DIMENSION(MAX_CASK_I)	IN	integer values
cask_r	INTEGER, DIMENSION(MAX_CASK_R)	IN	real values
cask_s	CHARACTER(LEN=STRLEN_MEDIUM), DIMENSION(MAX_CASK_S)	IN	string values

The minimum necessary information to define a new tracer in a tracer set is the `basename` of the tracer and the name of the `submodel` which defines the tracer. The `basename` must not contain any underscore ("_"). The `basename` can optionally be supplemented by a `subname`. The tracer within a set is identified by its unique `fullname`, which is the `basename`, if the `subname` is empty, or the `basename` followed by an underscore ("_") followed by the `subname`. The `unit` specifies the unit of the corresponding tracer *data*. The `longname` can be used for an extended description of the species. `medium`, `quantity` and `type` of the tracer can be specified, if the default values (AIR, AMOUNTFRACTION, SINGLE) are not appropriate. With the three containers `cask_i`, `cask_r`, and `cask_s` the additional *meta-information* can be specified, e.g., with the following sequence:

```
...
USE messy_main_tracer, ONLY: new_tracer, MAX_CASK_I, DEFAULT_CASK_I, OFF &
                           , MAX_CASK_R, DEFAULT_CASK_R, R_MOLARMASS &
                           , I_ADVECTION
...
```

```
INTEGER,  DIMENSION(MAX_CASK_I) :: MY_CASK_I
REAL(DP), DIMENSION(MAX_CASK_R) :: MY_CASK_R
...
MY_CASK_I(:) = DEFAULT_CASK_I(:)
MY_CASK_R(:) = DEFAULT_CASK_R(:)
MY_CASK_I(I_ADVECTION) = OFF
MY_CASK_R(R_MOLARMASS) = 30.0_dp
...
CALL new_tracer(status, setname, basename             &
               , cask_i=MY_CASK_I, cask_R=_MY_CASK_R)
...
```

The index `idx` of the tracer in the set can optionally be retrieved for further application with the subroutine `set_tracer`, and / or for addressing the corresponding tracer *data* memory.

3.2.2 The subroutine `set_tracer`

The subroutine `set_tracer` specifies the meta-information of a tracer in a tracer set.

`SUBROUTINE set_tracer`		`(status ,setname ,idx, flag , i | r | s)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
setname	CHARACTER(LEN=*)	IN	name of the tracer set
idx	INTEGER	IN	index of tracer in tracer set
flag	INTEGER	IN	container number
i*)	INTEGER	IN	integer container content
r*)	REAL(DP)	IN	real container content
s*)	CHARACTER(LEN=STRLEN_MEDIUM)	IN	string container content

*)Note: This subroutine is threefoldly overloaded for integer, real and string values respectively.

With each call to this subroutine one specific meta-information container can be filled. The corresponding tracer is identified by the name of the tracer set and the index of the tracer in the tracer set, e.g.:

```
...
USE messy_main_tracer, ONLY: set_tracer, R_MOLARMASS
...
CALL set_tracer(status, setname, idx, R_molarmass, 30.0_dp)
...
```

The index can be retrieved from the call to the subroutine `new_tracer`.

3.2.3 The subroutine `get_tracer`

The subroutine `get_tracer` retrieves information about a tracer in a tracer set.

`SUBROUTINE get_tracer`		`(status ,setname ,basename [,subname] [,idx] [,fullname] [,longname] [,unit] [,submodel] [,medium] [,quantity] [,type] [,trinfo] [,pxt] [,pxtm1] [,pxtte] [,pxmem])`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
setname	CHARACTER(LEN=*)	IN	name of the tracer set
basename	CHARACTER(LEN=*)	IN	basename of the tracer
optional arguments:			
subname	CHARACTER(LEN=*)	IN	subname of the tracer
idx	INTEGER	OUT	index of tracer in tracer set
fullname	CHARACTER(LEN=*)	OUT	fullname of the tracer
longname	CHARACTER(LEN=*)	OUT	string information about the tracer
unit	CHARACTER(LEN=*)	OUT	unit of the tracer *data*
submodel	CHARACTER(LEN=*)	OUT	name of submodel which defined the tracer
medium	INTEGER	OUT	medium of the tracer
quantity	INTEGER	OUT	quantity of the tracer
type	INTEGER	OUT	type of the tracer
trinfo	TYPE(t_trinfo)	OUT	*meta-information* structure
pxt	REAL(DP), DIMENSION(:,:,:)	POINTER	pointer to tracer memory (t)
pxtm1	REAL(DP), DIMENSION(:,:,:)	POINTER	pointer to tracer memory ($t - \Delta t$)
pxtte	REAL(DP), DIMENSION(:,:,:)	POINTER	pointer to tracer memory (tendency)
pxmem	REAL(DP), DIMENSION(:,:,:,:)	POINTER	pointer to additional memory instances

Knowing the `basename` (and optional `subname`) of the tracer, the *meta-information* of the tracer in the set can be retrieved and local pointers to the corresponding tracer *data* memory can be set. The `fullname` is the `basename`, if the `subname` is empty, or the `basename` followed by an underscore (“_”) and the `subname`. The structure `trinfo` contains (a copy of) the *meta-information* of the tracer (see Sect. 2). The following example shows how to access it:

```
...
USE messy_main_tracer, ONLY: get_tracer, t_trinfo, R_molarmass
...
TYPE(t_trinfo) :: ti
...
CALL get_tracer(status, setname, basename, trinfo=ti)
...
WRITE(*,*) "The molar mass is :",ti%info%meta%cask_R(R_molarmass)
...
```

The pointers `pxt`, `pxtm1` and `pxtte` point to the *instances* corresponding to the tracer *data* at model time step t, $t-\Delta t$, and to the tracer tendency, respectively, if the tracer set has been set up (subroutine `setup_tracer_set`) with three or more *instances* and `l_tfstd` = .TRUE.. In this case, `pxmem` points to all remaining *instances*, or causes `status` > 0, if only three *instances* are defined. If the tracer set has been setup in a different way, accessing `pxtm1` and / or `pxtte` will result in a `status` > 0. Accessing `pxmem`, if only one instance has been defined, will also result in `status` > 0.

It is highly recommended to test the `status` of this routine after it has been called.

3.2.4 The subroutine `get_tracer_list`

The subroutine `get_tracer_list` retrieves the tracer indices of all tracers with the same basename from a tracer set.

`SUBROUTINE get_tracer_list`		`(status ,setname ,basename ,idxs [,subnames])`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
setname	CHARACTER(LEN=*)	IN	name of the tracer set
basename	CHARACTER(LEN=*)	IN	basename of the tracer
idxs	INTEGER, DIMENSION(:)	POINTER	tracer indices
optional arguments:			
subnames	CHARACTER(LEN=STRLEN_MEDIUM), DIMENSION(:)	POINTER	tracer subnames

Optionally, the `subnames` of all tracers with the specified `basename` can be retrieved. The two pointers will be allocated with the number of tracers found (≥ 0).

3.2.5 The subroutine `tracer_iniflag`

The subroutine `tracer_iniflag` sets the initialisation flag.

`SUBROUTINE tracer_iniflag`		`(status ,setname ,id ,linit)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
setname	CHARACTER(LEN=*)	IN	name of the tracer set
id	INTEGER	IN	index of tracer in tracer set
linit	LOGICAL	IN	initialisation state

`linit`=.TRUE. means the tracer *data* is initialised.

3.2.6 The function `tracer_error_str`

The function `tracer_error_str` returns status information.

`CHARACTER(LEN=STRLEN_VLONG) FUNCTION tracer_error_str (status)`			
name	type	intent	description
mandatory arguments:			
status	INTEGER	IN	

The error status is converted into a meaningful message.

3.2.7 The function `param2string`

The function `param2string` converts a parameter to a string.

`CHARACTER(LEN=STRLEN_MEDIUM) FUNCTION param2string (i ,mode)`			
name	type	intent	description
mandatory arguments:			
i	INTEGER	IN	
mode	CHARACTER(LEN=*)	IN	

With this information tracer *meta-information* can be converted to strings, e.g., to generate readable output or attributes. Four `mode`s are available: 'switch', 'type', 'medium', and 'quantity'. An application sequence is for example:

```
...
CALL get_tracer(status, setname, basename, trinfo=ti)
str = param2string(ti%meta%ident%medium, 'medium')
! will return 'AIR', or 'AEROSOL', or ...
...
CALL get_tracer(status, setname, basename, trinfo=ti)
str = param2string(ti%meta%cask_i(I_ADVECTION), 'switch')
! will return 'ON', or 'OFF'
...
```

3.2.8 The subroutine `full2base_sub`

The subroutine `full2base_sub` converts a fullname to basename and subname.

`SUBROUTINE full2base_sub`		`(status ,fullname ,basename ,subname)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
fullname	CHARACTER(LEN=*)	IN	fullname of the tracer
basename	CHARACTER(LEN=*)	OUT	basename of the tracer
subname	CHARACTER(LEN=*)	OUT	subname of the tracer

4 A simple example application

A typical sequence of calls during the three phases of a model simulation is provided in `tracer_bml.f90`, an example basemodel (BML) and its base model interface layer (BMIL) `messy_main_tracer_bi.f90`. For the implementation of TRACER into a specific model, it is recommended to modify and use `messy_main_tracer_bi.f90`, since it contains also correct calls to the TRACER submodels TRACER_FAMILY and TRACER_PDEF, and further provides a high-level interface for the tracer-family conversion.

INITIALISATION PHASE

- `bml_initialize`: initialise base model
- `main_tracer_initialize`
  - `main_tracer_read_nml_ctrl`: read CTRL namelist and set switches
  - `main_tracer_family_initialize`:
    - `tracer_family_read_nml_ctrl`: read CTRL_FAMILY namelist and set switches
  - `main_tracer_pdef_initialize`:
    - `tracer_pdef_read_nml_ctrl`: read CTRL_PDEF namelist and set switches
- initialise MESSy submodels
- `main_tracer_new_tracer(1)`
  - `new_tracer_set`: define new tracer set(s)
- add tracers by MESSy submodels with `new_tracer`
- `main_tracer_new_tracer(2)`
  - `main_tracer_family_new_tracer`: define tracer families according to CTRL_FAMILY namelist
- `main_tracer_new_tracer(3)`
  - `print_tracer_set`: diagnostic output
- `main_tracer_init_memory(1)`
  - `setup_tracer_set`: fixate meta-information and allocate data memory
  - `get_tracer_set`: set pointers to tracer set data memory
- setup memory of MESSy submodels
- `main_tracer_init_memory(2)`
  - `main_tracer_pdef_init_mem`: additional memory for TRACER_PDEF
  - `main_tracer_family_init_mem`: set meta information of family-members to fraction

- `main_tracer_init_coupling`
  - `main_tracer_family_init_cpl`: reset meta information of family-members
  - `main_tracer_pdef_init_cpl`: fixate settings of TRACER_PDEF
- coupling of MESSy submodels with `get_tracer`
- `main_tracer_init_tracer(1)`: check initialisation after restart
- initialise tracers via MESSy submodels
- `main_tracer_init_tracer(2)`: check initialisation state; initialise with constant
- `main_tracer_init_tracer(3)`
  - `print_tracer_set_val`: diagnostic output

TIME INTEGRATION PHASE

- `main_tracer_global_start(1)`
  - `main_tracer_pdef_global_start`: set trigger
- global_start of MESSy submodels
- `main_tracer_global_start(2)`
  - `main_tracer_family_global_start`:
    - `tracfamily_2_sum`: conversion of type-2 families into family mode (summation)
    - `tracfamily_1_t2f`: conversion of type-1 families into family mode
- processes in family mode; e.g., advection
- `main_tracer_afteradv`
  - `main_tracer_family_afteradv`
    - `tracfamily_1_f2t`: conversion of type-1 families into tracer mode
    - `tracfamily_2_rsc`: conversion of type-2 families into tracer mode (rescaling)
- `main_tracer_fconv_glb`: optional family conversion
- ... START LOOP OVER OUTER RANK IN REPRESENTATION
- `main_tracer_local_start`: set pointers (reduced in rank) to data memory
- `main_tracer_fconv_loc`: optional family conversion
- ... END LOOP OVER OUTER RANK IN REPRESENTATION
- main_tracer_global_end
  - `tracpdef_airmass`: set tracer set (representation) specific airmass for global tracer mass integration
  - `main_tracer_pdef_global_end`: global tracer mass integration
- output of results

FINALISING PHASE

- `main_tracer_free_memory`
  - `clean_tracer_set`
  - `main_tracer_pdef_free_mem`
  - `main_tracer_family_free_mem`

5 The file `messy_main_tracer_family.f90`

5.1 The subroutine `tracer_family_read_nml_ctrl`

The subroutine `tracer_family_read_nml_ctrl` reads the CTRL_FAMILY namelist.

`SUBROUTINE tracer_family_read_nml_ctrl`		`(status ,iou)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
iou	INTEGER	IN	Fortran95 input unit

5.2 The subroutine `tracfamily_init`

The subroutine `tracfamily_init` processes the information read from the CTRL_FAMILY namelist.

`SUBROUTINE tracfamily_init`		`(status)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	

5.3 The subroutine `tracfamily_newtrac`

The subroutine `tracfamily_newtrac` defines new tracers (for the families) according to the CTRL_FAMILY namelist.

`SUBROUTINE tracfamily_newtrac`		`(status ,ldiagout)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
ldiagout	LOGICAL	IN	diagnostic output ?

The first valid tracer in a list of family members determines the *meta-information* of the family tracer. Tracers which are not defined, tracers which are already a member of another family (except for members of type-2 families without rescaling), empty families and families with the same name as already defined tracers are ignored.

5.4 The subroutine `tracfamily_initmode`

The subroutine `tracfamily_initmode` initialises the internal mode as tracer mode.

`SUBROUTINE tracfamily_initmode`	`()`

5.5 The subroutine `tracfamily_meta`

The subroutine `tracfamily_meta` converts the *meta-information* for type-1 families and their members between tracer mode and family mode in both directions.

`SUBROUTINE tracfamily_meta`		`(status ,flag ,callstr ,setname ,ldiagout)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
flag	INTEGER	IN	1: conversion into tracer mode; 2: conversion into family mode
callstr	CHARACTER(LEN=*)	IN	name of calling routine
setname	CHARACTER(LEN=*)	IN	name of the tracer set
ldiagout	LOGICAL	IN	diagnostic output ?

5.6 The subroutine `tracfamily_1_f2t`

The subroutine `tracfamily_1_f2t` converts type-1 families into the tracer mode (Eqs. (12)-(14)).

`SUBROUTINE tracfamily_1_f2t`		`(status ,callstr ,p_pe ,setname ,ztmst ,jjrow [,ksize] [,ltesubst])`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
callstr	CHARACTER(LEN=*)	IN	name of calling routine
p_pe	INTEGER	IN	*)
setname	CHARACTER(LEN=*)	IN	name of the tracer set
ztmst	REAL(DP)	IN	model time step length (Δt)
jjrow	INTEGER	IN	*)
optional arguments:			
ksize	INTEGER	IN	size of 1st data rank

*)Note: `p_pe` and `jjrow` are used for restricting diagnostic output (`l_verbose`=.TRUE. in CTRL_FAMILY namelist) to only one task (`i_diag_pe` in CTRL_FAMILY namelist) in a parallel environment and to only one row (`i_diag_jrow` in CTRL_FAMILY namelist) along the 3rd rank of the data representation.

5.7 The subroutine `tracfamily_1_t2f`

The subroutine `tracfamily_1_t2f` converts type-1 families into the family mode (Eqs. (3)-(5), (7)-(9) and (10)).

`SUBROUTINE tracfamily_1_t2f`		`(status ,callstr ,p_pe ,setname ,ztmst ,jjrow [,ksize] [,l_frac])`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
callstr	CHARACTER(LEN=*)	IN	name of calling routine
p_pe	INTEGER	IN	*)
setname	CHARACTER(LEN=*)	IN	name of the tracer set
ztmst	REAL(DP)	IN	model time step length (Δt)
jjrow	INTEGER	IN	*)
optional arguments:			
ksize	INTEGER	IN	size of 1st data rank
l_frac	LOGICAL	IN	calculate fractions ?

*)Note: `p_pe` and `jjrow` are used for restricting diagnostic output (`l_verbose`=.TRUE. in CTRL_FAMILY namelist) to only one task (`i_diag_pe` in CTRL_FAMILY namelist) in a parallel environment and to only one row (`i_diag_jrow` in CTRL_FAMILY namelist) along the 3rd rank of the data representation.

The optional switch `l_frac`=.FALSE. (default: .TRUE.) is used to restrict the conversion only to the summation of the families (i.e., Eqs. (3)-(5)), omitting the calculation of the fractions (Eqs. (7)-(9)) and the storage (Eq. (10)). This is used to update the family tracers just before the model output.

5.8 The subroutine `tracfamily_2_rsc`

The subroutine `tracfamily_2_rsc` converts type-2 families into the tracer mode (rescaling of the tracers, Eqs.(15)-(17)).

`SUBROUTINE tracfamily_2_rsc`		`(setname ,ztmst ,jjrow)`	
name	type	intent	description
mandatory arguments:			
setname	CHARACTER(LEN=*)	IN	name of the tracer set
ztmst	REAL(DP)	IN	model time step length (Δt)
jjrow	INTEGER	IN	row along 3rd rank of data

5.9 The subroutine `tracfamily_2_sum`

The subroutine `tracfamily_2_sum` converts type-2 families into the family mode (summation, Eqs. (3)-(5)).

`SUBROUTINE tracfamily_2_sum`		`(setname ,jjrow)`	
name	type	intent	description
mandatory arguments:			
setname	CHARACTER(LEN=*)	IN	name of the tracer set
jjrow	INTEGER	IN	row along 3rd rank of data

5.10 The subroutine `tracfamily_freemem`

The subroutine `tracfamily_freemem` deallocates the additional memory used to store information for tracer families.

`SUBROUTINE tracfamily_freemem`	`()`

6 The file `messy_main_tracer_pdef.f90`

6.1 The subroutine `tracer_pdef_read_nml_ctrl`

The subroutine `tracer_pdef_read_nml_ctrl` reads the CTRL_PDEF namelist.

`SUBROUTINE tracer_pdef_read_nml_ctrl`		`(status ,iou)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
iou	INTEGER	IN	Fortran95 input unit

6.2 The subroutine `tracpdef_initmem`

The subroutine `tracpdef_initmem` allocates and initialises the required memory.

`SUBROUTINE tracpdef_initmem`		`(nprocs)`	
name	type	intent	description
mandatory arguments:			
nprocs	INTEGER	IN	number of processors in parallel environment

6.3 The subroutine `tracpdef_settings`

The subroutine `tracpdef_settings` processes the information read from the CTRL_PDEF namelist.

`SUBROUTINE tracpdef_settings`		`(ldiagout)`	
name	type	intent	description
mandatory arguments:			
ldiagout	LOGICAL	IN	diagnostic output ?

6.4 The subroutine `tracpdef_airmass`

The subroutine `tracpdef_airmass` sets the airmass used for the tracer mass integration.

`SUBROUTINE tracpdef_airmass`		`(setname ,airmass)`	
name	type	intent	description
mandatory arguments:			
setname	CHARACTER(LEN=*)	IN	name of the tracer set
airmass$^{*)}$	REAL(DP), DIMENSION(:,:,:)	IN	airmass in kg
airmass$^{*)}$	REAL(DP)	IN	airmass in kg

$^{*)}$Note: This subroutine is overloaded for setting the airmass in the corresponding tracer set *representation* (e.g., the grid) either variable along the *representation* dimensions, or constant in all points of the corresponding tracer set *representation*.

6.5 The subroutine `tracpdef_integrate`

The subroutine `tracpdef_integrate` calculates the global tracer masses (Eqs. (18) and (19)) and checks the tolerance criterium for the negative mass (Eq. (20)).

`SUBROUTINE tracpdef_integrate`		`(status ,flag ,time_step_len ,p_pe)`	
name	type	intent	description
mandatory arguments:			
status	INTEGER	OUT	
flag	INTEGER	IN	switch (1 or 2)
time_step_len	REAL(DP)	IN	time step length (Δt)
p_pe	INTEGER	IN	number of process in parallel environment

This routine needs to be called twice. Once with `flag`=1 for the integration (summation) on each processor in the parallel environment. After this, the results of all processors need to be distributed to each other. Then this routine is called a second time (with `flag`=2) for the integration (summation) over all processors and the checking of the tolerance criterion.

6.6 The subroutine `tracpdef_freemem`

The subroutine `tracpdef_freemem` deallocates the memory used for TRACER_PDEF.

`SUBROUTINE tracpdef_freemem`	`()`

6.7 The subroutine `tracpdef_print`

The subroutine `tracpdef_print` outputs the global tracer masses of all tracer sets.

`SUBROUTINE tracpdef_print`		`(ldiagout)`	
name	type	intent	description
mandatory arguments:			
ldiagout	LOGICAL	IN	switch

The switch `ldiagout` can be used to restrict the output to one processor in a parallel environment.