ECHAM6 Boundary Condition Scheme

Martin Schultz, Sabine Schröder, Sebastian Rast, Hauke Schmidt, Olaf Stein
July 2012

Minor update: September 2012 (EF_TIMEOFFSET)

1. Motivation

The atmospheric model ECHAM6 requires several “external” boundary conditions, which may vary depending on the scope of the simulation (e.g. nudging, coupled ocean, coupled chemistry, etc.). In former ECHAM versions there was no standardized way to define or prescribe these boundary conditions. Therefore, several different routines existed which all require somewhat different calls and input (i.e. file definitions), and this makes it hard to maintain, extend and explain the code to new users. Based on the immediate need for chemistry modelling with ECHAM6 we therefore designed a new scheme to handle boundary conditions in ECHAM6 and its submodels in a coherent, flexible and efficient manner.

There are several different types of boundary conditions, and the new scheme attempts to structure these into a limited number of categories which can largely be handled with the same software code. The scheme takes the following aspects of boundary conditions into account:

- **type:** for some boundary conditions model routines may exist which compute these conditions interactively, but the user may alternatively wish to prescribe the values of a boundary condition through input from a file or through the definition of constant values or even to switch off the whole condition. A change from one mode to the other should not require code modifications.
  See EF_TYPE structure tag
- **dimensionality:** many boundary conditions are 2-dimensional fields and they may either describe a field in lon-lat dimensions (surface) or as zonal mean values (lat-lev dimensions). In some cases boundary conditions may also be defined in 3 dimensions.
  See EF_GEOMETRY, EF_MINLEV, EF_MAXLEV, BC_DOMAIN, BC_ALTITUDE, BC_LEVEL structure tags
- **physical meaning:** boundary conditions can often be specified as value or flux conditions which may require different treatment in the code. This is up to the user to decide
  See documentation of subroutine BC_APPLY
- **application mode:** the simplest way is to use boundary conditions as a simple substitution of model values. However, in some cases one prefers to add external values to an existing field or apply a relaxation term to gently nudge the model towards the boundary condition values. In this case, a relaxation time must be specified.
  See BC_MODE and BC_RELAXTIME structure tags
- **time handling:** many boundary conditions are provided as monthly values, but more and more data become available that allow for a finer time resolution. Furthermore the user may wish to decide whether the field should be interpolated in time or simply be used as a step function. Another distinction is between “climatological” boundary conditions which don’t change between years and “real time” boundary conditions, which define the variables for a specific point in time. Furthermore, the user may wish to apply a time offset to a given boundary condition value for sensitivity studies or to generate ensembles, or the user wants to select a specific record in a file.
  See EF_TIMEDEF, EF_TIMEOFFSET, EF_TIMEINDEX, EF_INTERPOLATE structure tags
- **resolution:** often, a boundary condition file is not provided in the model resolution but needs to be interpolated before use. It is the ECHAM6 philosophy that this is done externally, i.e. prior to the model simulation. Therefore, the boundary condition

scheme does not include any horizontal interpolation routines. Vertical interpolation (e.g. from altitude or pressure levels) will be done if needed (for example for aircraft emissions).

In spite of these different aspects that must be considered, the scheme aims at computational efficiency and simplicity with respect to the user interface. Another important aspect to be considered is the possibility for rigorous error checking and some assistance for diagnosing what values were actually provided to the model.

2. Main Elements

The boundary condition scheme consists of two modules plus an extension for easier handling of multi-species and multi-sector emission data:

mo_external_field_processor
mo_boundary_condition
mo_emi_matrix

Mo_external_field_processor takes care of all aspects of getting the boundary condition value from the external source if EF_TYPE specifies that an external source is used. In particular, the module contains everything that is necessary to obtain data from netcdf files which are the standard file format for boundary conditions. No user code is required to select a new record from an input file or open the next file if time-dependent boundary conditions are spread across separate files. Mo_external_field_processor contains the low-level machinery for the boundary condition scheme and is generally not used in user code, except to obtain the values of predefined constants for the EF_ structure tags in the bc_nml structure (see below).

Mo_boundary_condition is the main user interface and contains additional routines to centrally manage the list of boundary conditions and make sure they are regularly updated and passed to the user in the form that they are expected. The user "communicates" with the boundary condition scheme via a set of seven routines in mo_boundary_condition:

bc_define
bc_set
bc_apply
bc_find
bc_query
bc_modify
p_bcast_bc

Another routine, which is used internally, but may be useful to know about is:

bc_list_read (integrated in stepon)

In order to implement a boundary condition, the user must define a variable of type bc_nml to specify the boundary condition characteristics and store the return value from bc_define as an integer handler to identify the boundary condition in bc_apply. If EF_TYPE is set to EF_MODULE, no boundary values will be given by the mo_boundary_condition-module, but instead the user must use the bc_set routine to store the results from an interactive parameterisation for later use as a boundary condition elsewhere in the code. P_bcast_bc is needed when the user allows to modify default values of the bc_nml structure in a namelist. In this case, p_bcast_bc must be called after the namelist read and before bc_define is called so that all processors define the boundary condition in the same way.

Mo_emi_matrix implements an extension to mo_boundary_condition in order to simplify handling of multiple emission data sets for multiple species and improve documentation of which emissions where actually used in a given model run. Mo_emi_matrix is described in section 6 below.

3. The bc_nml structure

The bc_nml structure is defined in mo_boundary_condition and provides the user the possibility to define the characteristics of the boundary condition. A variable of this type must be passed as main element to the bc_define routine in order to set-up a new boundary condition. Note that the user is responsible to exclude “misuse” of a boundary condition when he/she allows for the modification of bc_nml values via a namelist. For example, if no interactive parameterisation exists for a specific boundary condition, the user code should return an error message if the EF_TYPE tag is set to EF_MODULE.

Here, we list the structure definition of the bc_nml type and the following subsections contain further documentation on the individual tags.

```
  TYPE, PUBLIC :: bc_nml
  ! external field properties:
    ! source of the boundary condition
    INTEGER               :: ef_type = EF_INACTIVE
                             ! possible values:
                             ! EF_INACTIVE
                             ! EF_VALUE
                             ! EF_FILE
                             ! EF_MODULE
    ! information for type=ef_file
    ! template for filename. Example: bc_ch4.%Y4.%T0.nc
    CHARACTER(LEN=512)    :: ef_template = ''
    ! variable name in file
    CHARACTER(LEN=512)    :: ef_varname= ''
    ! geometry of the file variable
    INTEGER               :: ef_geometry = EF_UNDEFINED
                             ! possible values:
                             ! EF_UNDEFINED  ! only for internal use
                             ! EF_3D
                             ! EF_LONLAT
                             ! EF_LATLEV
                             ! EF_LEV
                             ! EF_LAT
                             ! EF_SINGLE
    ! definition of the time values in the file
    INTEGER               :: ef_timedef = EF_TIMERESOLVED
                             ! possible values:
                             ! EF_TIMERESOLVED
                             ! EF_IGNOREYEAR
                             ! EF_CONSTANT
    ! offset to time values (e.g. to shift from mid-month to 1st day)
    REAL(dp)              :: ef_timeoffset = 0.0_dp
                             ! (unit: unit of time record in input file)
    ! fixed record number to select a specific time in the file
    ! (if ef_timedef = EF_CONSTANT)
    INTEGER               :: ef_timeindex = -1
    ! time interpolation
    INTEGER               :: ef_interpolate = EF_NOINTER
                             ! possible values:
                             ! EF_NOINTER
                             ! EF_LINEAR
                             ! EF_CUBIC -- not yet possible in this version
    ! scaling factor
    REAL(dp)              :: ef_factor = 1.0_dp
    ! expected unit string in netcdf file
    CHARACTER(LEN=30)     :: ef_units = ''
    ! information for type=ef_value
    ! globally uniform boundary condition value
    REAL(dp)              :: ef_value = 0.0_dp
  ! define application of boundary condition
    ! (vertical) domain where field should be applied
    INTEGER              :: bc_domain = BC_EVERYWHERE
                             ! possible values:
                             ! BC_EVERYWHERE
                             ! BC_BOTTOM
                             ! BC_TOP
                             ! BC_LEVEL
                             ! BC_ALTITUDE
                             ! BC_PRESSURE
    ! minimum and maximum model level where field shall be applied
    INTEGER              :: bc_minlev       = -1
    INTEGER              :: bc_maxlev       = -1
    ! mode of application
    INTEGER              :: bc_mode         = BC_REPLACE
                             ! possible values:
                             ! BC_REPLACE
                             ! BC_ADD
                             ! BC_RELAX
                             ! BC_SPECIAL
    ! relaxation time for mode = bc_relax
    REAL(dp)             :: bc_relaxtime    = 0._dp
  END TYPE bc_nml
```

3.1 EF_TYPE

The EF_TYPE tag determines where the boundary condition will be obtained from. Its four possible values are EF_INACTIVE, EF_VALUE, EF_FILE and EF_MODULE.

If the boundary condition is de-activated by setting EF_TYPE to EF_INACTIVE, then the bc_define routine will not enter anything in its internal list structure and it will not allocate any memory. The return value ibc from bc_define will be -1. User code should always test for the validity of the boundary condition index (ibc > 0) before calling bc_apply to set boundary condition values.

EF_TYPE = EF_VALUE allows the definition of a constant value as a boundary condition. The value itself has to be stored in EF_VALUE, the default value is 0._dp. Depending on the dimensionality of the boundary condition (see below), the constant value will be applied to all geographical locations of the 2D or 3D field (as defined by bc_domain and/or bc_level) which the user supplies to the bc_apply routine.

Most of the other EF_x tags are only relevant when EF_TYPE is set to EF_FILE, because in this case a lot more information is required in order to obtain the right data in the correct format. The boundary condition scheme only accepts netcdf files as input, and these must adhere to a few very general rules (see Annex 1). At a minimum, the EF_TEMPLATE, EF_VARNAME and EF_GEOMETRY tags must be set, before a file-type boundary condition can be used. If no further EF_x tags are modified, the boundary condition scheme will then expect a time-dependent boundary condition, where the time variable in the file indicates the first occurrence of the boundary condition values from that record. When the model time step passes the date and time of the current record, the next suitable time record will be determined and when the end of the file is reached, the boundary condition scheme will look for the next suitable file. No scaling and no interpolation will be applied to the values that are read from the file. These characteristics can be controlled with the remaining EF_x tags described below.

If EF_TYPE is set to EF_MODULE, no boundary values will be given by the mo_boundary_condition-module, but instead the user must use the bc_set routine to store the results from an interactive parameterisation for later use as a boundary condition elsewhere in the code.

3.2 EF_TEMPLATE

EF_TEMPLATE holds the path and name of the boundary condition netcdf data file. The template can contain specific tokens which will be replaced by the current date, the model resolution, etc. at runtime (see mo_filename). Specific tokens supported by the boundary condition scheme are:

%Y0, %y0	: will be replaced by the year of the simulation
%M0, %m0	: will be replaced by the month (as number) of the simulation
%A0, %a0	: will be replaced by the month (as short name (like Jan, Feb, Mar)) of the simulation
%D0, %d0	: will be replaced by the day of the simulation
%H0, %h0	: will be replaced by the hour of the simulation
%I0, %i0	: will be replaced by the minute of the simulation
%S0, %s0	: will be replaced by the second of the simulation
%C0, %c0	: will be replaced by the actual species (see mo_emi_matrix)
%T0	: will be replaced by the horizontal model resolution (spectral truncation) in the format of a string such as ‘T42’
%L0	: will be replaced by the vertical model resolution (number of levels) in the format of a string such as ‘L39’

Every token can be additionally given a specific length (e.g. %Y4 results the year to be expanded to 4 digits). If length ‘0’ is specified then the smallest field to hold the entry is chosen.

3.3 EF_VARNAME

This tag defines the case-sensitive variable name that shall be read from the file which is given by EF_TEMPLATE. If this variable is not found in the file, an error will occur.

3.4 EF_GEOMETRY

The boundary condition scheme always provides either a 2D field (1:nproma, 1:ngblks) or a 3D field(1:nproma, 1:klev, 1:ngblks) in bc_apply. However, it allows for great flexibility in terms of how the boundary condition values are stored in the external file. While the normal case would likely be to provide 3D files for 3D boundary conditions and 2D EF_LONLAT files for 2D boundary conditions, you can also provide a time-dependent, globally uniform value (EF_SINGLE, for example for greenhouse gas concentrations in standard ECHAM runs) or values from a zonal mean climatology (EF_LATLEV or EF_LAT). It is also possible to give individual values for each vertical level which are then spread uniformly in lon-lat directions (EF_LEV).

Note that the external field geometry (EF_GEOMETRY) really only concerns the geometry of the input file and it does not control in any way where the boundary condition will eventually be applied in bc_apply (see BC_DOMAIN below). Nevertheless, there are a few restrictions:

The variable must not have additional dimensions as it's supposed to have by the values of EF_GEOMETRY and EF_TIMEDEF (see below) (e.g. if a 2D boundary condition is expected, there should be no vertical dimension for this variable),

As no horizontal interpolation is performed in ECHAM at runtime, the lon and lat dimensions of the input file must be consistent with the model grid,

Unless the boundary condition scheme knows that it must interpolate vertically from a pressure or altitude grid onto the model levels (see BC_DOMAIN below), the number of vertical levels in the file must correspond to the number of vertical levels in the model.

If EF_GEOMETRY is set to EF_UNDEFINED (the default value) and the code is not able to determine the geometry value automatically from the input file, the code will report an error when it attempts to retrieve data from the input file.

All variables that are to be used as boundary condition may contain a time dimension in addition to the spatial dimensions given in the EF_GEOMETRY value.

3.5 EF_TIMEDEF

The EF_TIMEDEF tag determines if the input data in the netcdf file is to be interpreted as time-dependent data, climatological data or as constant values. The default is EF_TIMERESOLVED. In this case the code will always attempt to find the input record which best matches the current model time step. For this reason the time values must be strictly monotonic increasing and of the type "day(s) since" or "month(s) since" (more units to be added in future versions if needed), the "calendar"-attribute is always expected to be "standard" (="gregorian") (and is not interpreted by the boundary condition scheme if set to any other value). The time interval of the input files must at least cover the time of the beginning of the simulation. If the first time step of the model calculation is not covered by the input data, an error will occur and the model will stop. If the time interval of the input data stops before the simulation stops the input data of the last time step will be used until the end of the simulation. During the run, the input data from the record is used whose time value is equal or prior to the simulation time. When the simulation time passes the time value of the next record in the input file, this record will become the record whose data is used.

You can change the perceived data of the input data by setting an EF_TIMEOFFSET <> 0 (see below).

EF_IGNOREYEAR behaves similarly, but as the name suggests, the current year of the simulation is ignored, so that the input data is treated as climatology. This is typically used with monthly mean climatological files. Note that the actual choice of the input file can still depend on the EF_TEMPLATE value. If EF_TEMPLATE contains for example a %Y4 token, the boundary condition scheme will replace this token by the current model year and open the respective file.

When EF_TIMEDEF is set to EF_CONSTANT, the boundary condition scheme expects a value > 0 in the EF_TIMEINDEX tag, which then specifies the data record to be read. EF_TIMEINDEX must be <= the maximum number of records in the input file.

3.6 EF_TIMEOFFSET

This tag allows for shifting the dates given in the input file without having to edit the file itself. EF_TIMEOFFSET is provided in the actual time units of the input file and is applied to the time values before they are interpreted. A typical example where EF_TIMEOFFSET will be used are monthly mean files where the time value marks the centre of the month, but the boundary condition shall be applied from the start of the month. If the time units in the file are “day(s) since …”, an EF_TIMEOFFSET of -15._dp will provide a reasonable correction for the file dates, although it will not yield a perfect solution because of the different number of days in each month. Therefore it is strongly advised to prepare boundary condition files with time records that indicate exactly the starting time of each data record rather than relying on the corrective behaviour of EF_TIMEOFFSET.

Exempt from this warning are monthly mean emission files for biomass burning emissions. While the monthly records in these files have the same meaning as those in anthropogenic emission files (i.e. they cover monthly averages), they should be applied differently in the model, because their values are typically based on fire observations between the first and last day of the month. Hence, the emission fluxes for January should be applied from January 1st onwards, and not beginning on January 15th, as the file date would suggest. Here, one should indeed specify an EF_TIMEOFFSET = -15.dp. Anthropogenic emission files also have date labels in the middle of the month, but here the exact point in time is more like a “projection” onto this date. It is therefore recommended to use linear time interpolation (see EF_INTERPOLATE below) and not apply an EF_TIMEOFFSET.

3.7 EF_TIMEINDEX

This tag is only used if the EF_TIMEDEF tag is set to EF_CONSTANT. It then gives the index of the data record to be read from the file. EF_TIMEINDEX defaults to zero and must be set to a value between 1 and the number of records in the file if EF_TIMEDEF = EF_CONSTANT.

3.8 EF_INTERPOLATE

This tag is used if boundary conditions should be automatically interpolated in time, but only if EF_TIMEDEF is either set to TIME_RESOLVED or to IGNORE_YEAR. The default is to apply no interpolation (EF_NOINTER) and replace the actual values of a boundary condition whenever a new time record is available. Linear interpolation is used if EF_INTERPOLATE is set to EF_LINEAR (in future versions of the boundary condition scheme there will be a cubic interpolation available by setting EF_INTERPOLATE to EF_CUBIC – in this version EF_INTERPOLATE=EF_CUBIC will lead to linear interpolation).

3.9 EF_FACTOR

This tag is used to automatically scale values that are read from file. If EF_TYPE is not set to EF_FILE and EF_FACTOR is set by the user, it will be ignored and the user will get a warning message.

3.10 EF_UNITS

The EF_UNITS tag is intended to solve problems appearing quite often by confusions regarding the units of a variable in the input file and the units implicitly expected by the program code.
By explicitly setting the EF_UNITS tag to the units of the input variable, the program might be able to reject a given input file (when the explicitly given units don't match the ones given in the units-attribute of the input file), or at least be able to know about the real units of a given input file if the variables in the file have no units attribute at all (which is really bad practice anyway).
By this approach it can be assured that the model knows about the units of all input boundary conditions. The program might then be able to do a unit conversion if possible (when the units of the input file don't match the units implicitly expected by the program code) or stop the execution.
The details of this logic still need to be defined. Some suggestions:

- if EF_UNITS is not explicitly set by the user the code will still test if the units in the file (EF_TYPE=EF_FILE) given by the units attribute are what is expected. In some cases automatic unit conversion may be performed.
- if EF_UNITS is set the code shall test whether the file variables actually carry this units attribute (EF_TYPE=EF_FILE). If the variables in the file have no units attribute ef_units shall be treated as such. If EF_TYPE is not set to EF_FILE the code will test if the unit is the one implicitly expected.

In this version of the boundary condition scheme the EF_UNITS topic is founded but not implemented yet. Therefore setting the EF_UNITS tag will not have any effect and will in particular not lead to any checking of the input data.

3.11 EF_VALUE

The EF_VALUE tag allows using a boundary condition of a constant value. This tag is equally named to the named constant EF_VALUE which is a possible value of the EF_TYPE tag to indicate that this value is only active if EF_TYPE is set to EF_VALUE. The value itself has to be stored in the EF_VALUE tag, the default value is 0._dp. Depending on the dimensionality of the boundary condition (see below), the constant value will be applied to all geographical locations of the 2D or 3D field (as defined by BC_DOMAIN and/or BC_LEVEL).

3.12 BC_DOMAIN

The BC_DOMAIN tag determines where to apply the boundary condition. In order to minimize the dependency on a specific model resolution (number of levels), the 'BC_TOP' and BC_BOTTOM settings can be applied. BC_LEVEL, BC_PRESSURE or BC_ALTITUDE indicates a range of levels (see below: bc_minlev and bc_maxlev). If the boundary condition is 3-dimensional and bc_domain is set to BC_LEVEL, BC_PRESSURE or BC_ALTITUDE, then the dimensionality must agree with the specified domain. BC_DOMAIN=BC_PRESSURE or BC_ALTITUDE indicates that vertical interpolation of the boundary condition is necessary. BC_DOMAIN=BC_EVERYWHERE implies that the boundary condition will be applied to all levels of the given boundary condition field.

3.13 BC_MINLEV and BC_MAXLEV

If BC_DOMAIN is set to BC_LEVEL, BC_PRESSURE or BC_ALTITUDE a range of levels where the boundary condition should be applied to is defined via BC_MINLEV and BC_MAXLEV (only one level if BC_MINLEV=BC_MAXLEV). BC_LEVEL indicates that the given levels are to be interpreted as the indices of the model levels, BC_PRESSURE indicates that the values of BC_MINLEV and BC_MAXLEV are given in “Pa” (Pascal) and BC_ALTITUDE indicates the values of BC_MINLEV and BC_MAXLEV being in “m” (metres). The use of BC_MINLEV and BC_MAXLEV in combination with BC_PRESSURE and BC_ALTITUDE is not available in this version of the boundary condition scheme (but will be implemented in the next version).

3.14 BC_MODE

The bc_mode tag defines how the boundary condition should be applied to the model.
The default mode is BC_REPLACE which means that the model variable is replaced by the values of the boundary condition. BC_ADD is self-explanatory. If BC_MODE is set to BC_RELAX (not implemented in this version of the boundary condition scheme), a relaxation time must be provided. If BC_MODE is set to BC_SPECIAL, then the code will take care of applying the boundary condition properly. The user must take care of initializing, respectively setting all boundary conditions properly before calling BC_APPLY, when BC_MODE is set to BC_RELAX or BC_ADD.

3.15 BC_RELAXTIME

If BC_MODE is set to BC_RELAX, the relaxation time (in seconds) is defined via this tag. BC_RELAXTIME=0 leads to the same behaviour as BC_MODE=BC_REPLACE. This feature is not implemented yet in this version of the boundary condition scheme.

4. Description of the public routines

4.1 BC_DEFINE

This function is needed when a boundary condition is defined. The definition of a boundary condition is done via the bc_nml structure. It must be named and its BC_DOMAIN-dimensions must be given at definition time (as a 2D field read from file might be applied to a 2D or a 3D boundary condition in the model). The function returns an identifier (integer value) to this boundary condition.

The optional `lverbose` argument informs the user of all properties of the defined boundary condition in the log file.

4.2 BC_SET

The BC_SET routine is used to set the values of a special boundary condition (see above: BC_MODE=BC_SPECIAL). This boundary condition must have been previously defined by a call to BC_DEFINE. BC_SET is called with the integer handler of the boundary condition and the values to be set. An example of using BC_SET is given in chapter 5.4 MOZ_UVALBEDO.

4.3 BC_APPLY

The BC_APPLY routine will apply the boundary condition to the given field according to its previous definition by BC_DEFINE (see chapter 3 The bc_nml structure). BC_SET is called with the integer handler of the boundary condition and the field to receive the boundary condition values.

4.4 BC_FIND

The BC_FIND routine will find the index of a boundary condition given its name.

4.5 BC_QUERY

The BC_QUERY routine will find the type (see EF_TYPE) and/or the name of a boundary condition given its index.

4.6 BC_MODIFY

The BC_MODIFY routine allows the code to modify the name, type (see EF_TYPE), domain (see BC_DOMAIN) and/or ndims (see BC_DOMAIN) property of a boundary condition during the run time of the model.

4.7 P_BCAST_BC

This routine is needed when a boundary condition structure is set from a submodel (or an ECHAM) namelist. The bc_nml structure was designed specifically for this purpose. If it is read from a namelist, this will happen on the I/O processor only, and you must communicate the bc_nml settings to all the other processors before calling BC_DEFINE. As bc_nml is a structure, the generic P_BCAST interface cannot handle its transmission. Therefore, you must call P_BCAST_BC(bc_struc, p_io) instead of P_BCAST(value, p_io).

4.8 BC_LIST_READ

This routine is implemented in the main time loop (stepon) and ensures that boundary conditions are always kept up-to-date. There is no direct user interaction in this process, but it may help to understand at least vaguely what happens in the bc_list_read routine in order to avoid errors when defining new boundary conditions. Roughly telling, BC_LIST_READ loops over all boundary conditions, reads the next record of a boundary condition, when ever the simulation time passes the time value of this record in the input file, expands the new field to a suitable 2D or 3D field (if needed) and stores the data in a data stream for later calls of BC_APPLY.

5. Example applications

Three examples below demonstrate various aspects of the boundary condition scheme and show the user-friendliness of this concept. Note, that many changes with respect to the way the external field is retrieved or applied can be controlled simply by changing the structure elements upon initialisation with BC_DEFINE, so no code changes are necessary.

5.1 Simple 2D boundary condition using a constant value

Assume, we want to apply a zero-concentration boundary condition at the topmost model level for all tracers. Since all tracers thus apply the same value, only one boundary condition needs to be defined. Nevertheless, if you think you may ever wish to replace this boundary condition by a different one, which for example reads tracer-specific values from a file, then you should define an individual boundary condition for each tracer as follows. This must be done in your submodel initialisation module. The integer handler returned from BC_DEFINE must be publically accessible if you apply the boundary condition in a different module from the initialisation.

```
MODULE mo_my_bc
USE mo_kind,                 ONLY : dp
IMPLICIT NONE
PRIVATE
! PUBLIC : ibc_first    ! in this example we handle everything within
                        ! one module, thus ibc_first can be private.
INTEGER :: ibc_first = -1

SUBROUTINE my_init
USE mo_boundary_condition,       ONLY : bc_nml, bc_define, &
                                        BC_REPLACE, BC_TOP
USE mo_tracer,                   ONLY : ntrac
USE mo_external_field_processor, ONLY: EF_VALUE, EF_CONSTANT


TYPE (bc_nml)       :: mybc
INTEGER             :: idum, jt, ndims
CHARACTER (len=32)  :: c

!... your general initialisation stuff
!... set up the bc structure
  mybc%ef_type = EF_VALUE
  mybc%ef_timedef = EF_CONSTANT
  mybc%ef_value = 0._dp
  mybc%bc_mode = BC_REPLACE
  mybc%bc_domain = BC_TOP
  ndims = 2 ! the bc is only applied to the top level (BC_TOP)
!... you might want to set additional default values
```

```
!... loop over tracers and define one boundary condition for each
!    remember the return value from the first call
  DO jt = 1, ntrac
    WRITE (c, *) 'MYBC', jt
    idum = bc_define(c, mybc, ndims)
    IF (jt == 1) ibc_first = idum
  END DO
END SUBROUTINE my_init

SUBROUTINE zero_ub (kproma, kbdim, krow, pxt)
! Set tracer concentrations to boundary condition in upper level
USE mo_tracer,             ONLY : ntrac
USE mo_control,            ONLY : nlev
USE mo_boundary_condition, ONLY : bc_apply

INTEGER,  INTENT(in)    :: kproma   ! geographic block number of locations
INTEGER,  INTENT(in)    :: kbdim   ! geographic block maximum number of
                                   ! locations
INTEGER,  INTENT(in)    :: krow    ! geographic block number
REAL(dp), INTENT(inout) :: pxt(kbdim,nlev,ntrac)

INTEGER                  :: jt

  DO jt = 1, ntrac
    CALL bc_apply(ibc_first+jt-1, kproma, krow, pxt(:,:,jt)
  END DO

END MODULE mo_my_bc
```

5.2 Time-varying boundary condition which is updated from external files during the run

```
MODULE mo_my_bc
USE mo_kind,                 ONLY : dp
IMPLICIT NONE
PRIVATE

INTEGER            :: ibc

SUBROUTINE my_init
USE mo_boundary_condition,       ONLY : bc_nml, bc_define, &
                                        BC_REPLACE, BC_TOP
USE mo_external_field_processor, ONLY: EF_FILE, EF_TIMERESOLVED

TYPE (bc_nml) :: mybc
INTEGER       :: ndims


!... your general initialisation stuff
!... set up the bc structure
  mybc%ef_type = EF_FILE
  mybc%ef_template = "ch4_surface_1800-2100_%Y4_%T0.nc"
  mybc%ef_timedef = EF_TIMERESOLVED
  mybc%bc_mode = BC_REPLACE
  mybc%bc_domain = BC_TOP
  ndims = 2 ! the bc is only applied to the top level (BC_TOP)
!... you might want to set additional default values in this
!    BC is changed to a different type in other runs

  ibc = bc_define('myTimeBC', mybc, ndims)
END SUBROUTINE my_init
```

```
SUBROUTINE time_ub (kproma, kbdim, krow, jt, pxt)
! Set boundary condition in upper level to values read from file
USE mo_tracer,             ONLY : ntrac
USE mo_control,            ONLY : nlev
USE mo_boundary_condition, ONLY : bc_apply

INTEGER,  INTENT(in)    :: kproma   ! geographic block number of locations
INTEGER,  INTENT(in)    :: kbdim   ! geographic block maximum number of
                                   ! locations
INTEGER,  INTENT(in)    :: krow    ! geographic block number
INTEGER,  INTENT(in)    :: jt
REAL(dp), INTENT(inout) :: pxt(kbdim,nlev,ntrac)

CALL bc_apply(ibc, kproma, krow, pxt(:,:,jt)
END MODULE mo_my_bc
```

5.3 Three-dimensional boundary condition applied with a relaxation time

This example will be added in a later version of this documentation (at the time, when the relaxation factor is implemented in the boundary condition scheme).

5.4 MOZ_UVALBEDO: example for a module-derived boundary condition

```
MODULE mo_moz_uvalbedo

  USE mo_kind,               ONLY: dp

  IMPLICIT NONE

  PRIVATE

  SAVE

! module routines
  PUBLIC :: moz_uvalbedo_init        ! define boundary conditions
  PUBLIC :: moz_uvalbedo   ! compute combined UV albedo and store as
boundary condition

  PUBLIC :: ibc_uvalbedo             ! index of UV albedo boundary
condition.
                                     ! ibc_uvalbedo+1 is used for green map
                                     ! ibc_uvalbedo+2 is used for white map

! Variable declarations
  INTEGER            :: ibc_uvalbedo

  CONTAINS

SUBROUTINE moz_uvalbedo_init (uvalbedo_file)

  USE mo_boundary_condition, ONLY: bc_nml, bc_define, BC_EVERYWHERE,
BC_BOTTOM, BC_REPLACE, BC_SPECIAL, BC_RELAX
  USE mo_external_field_processor, ONLY: EF_FILE, EF_MODULE, EF_LONLAT,
EF_IGNOREYEAR

  CHARACTER (LEN=*), INTENT(in)       :: uvalbedo_file     ! filename from
mozctl namelist
```

```
  TYPE (bc_nml)                        :: bc_struc
  INTEGER                              :: idum


  ! set parameters for UV albedo boundary condition
  bc_struc%ef_type      = EF_MODULE
  bc_struc%ef_geometry  = EF_LONLAT
  bc_struc%bc_mode      = BC_SPECIAL
  bc_struc%bc_domain    = BC_EVERYWHERE
  ! define UV albedo boundary condition
  ibc_uvalbedo = bc_define('MOZART UV albedo', bc_struc,2,.TRUE.)

  ! set parameters for the green map
  bc_struc%ef_type      = EF_FILE
  bc_struc%ef_template  = TRIM(uvalbedo_file)
  bc_struc%ef_varname   = 'agreen'
  bc_struc%ef_geometry  = EF_LONLAT
  bc_struc%ef_timedef   = EF_IGNOREYEAR
  bc_struc%bc_mode      = BC_REPLACE
  idum = bc_define('MOZART snow-free UV albedo', bc_struc,2,.TRUE.)

  ! set parameters for the white map
  bc_struc%ef_varname   = 'awhite'
  idum = bc_define('MOZART snow-covered UV albedo', bc_struc,2,.TRUE.)

END SUBROUTINE moz_uvalbedo_init


SUBROUTINE moz_uvalbedo (kproma, krow, psnowcover)

  USE mo_boundary_condition,   ONLY: bc_set, bc_apply

  INTEGER, INTENT(in)       :: kproma, krow
  REAL(dp), INTENT(in)      :: psnowcover(:)

  REAL(dp)                  :: zgreen(kproma), zwhite(kproma), zalb(kproma)
  ! get values for green and white albedo
  CALL bc_apply(ibc_uvalbedo+1, kproma, krow, zgreen)
  CALL bc_apply(ibc_uvalbedo+2, kproma, krow, zwhite)
  ! compute combined UV albedo
  zalb = zgreen
  WHERE (psnowcover > 0.008_dp)  zalb = zwhite
  CALL bc_set(ibc_uvalbedo, krow,zalb)

END SUBROUTINE moz_uvalbedo
END MODULE mo_moz_uvalbedo
```

6. Emissions

A new concept in ECHAM6-HAMMOZ is the central control of all emissions via the **emi_spec.txt** file in the run directory. The advantages of this approach are not only a generic handling of 2D and 3D fluxes with sector and project specific handling possibilities, but also a clearly arranged input file, which documents all emission definitions at one glance.

The directory where all individual emission files can be found is defined by the entry **emi_basepath** of namelist **submodelctl**; the emi_spec.txt file can by this manner be used on different machines with different file systems without any changes.

Emissions can also be declared as interactive by specifying EF_MODULE as external field type.

emi_spec.txt defines two fundamental properties common to all emissions:

1. **SECTORS**: In order to maximize consistency among aerosol and gas-phase emissions, a uniform sector treatment has been introduced. All emissions from one sector are treated alike (for example: all traffic emissions will be released at the surface, all emissions from power generation will be released at the second-lowest model level).
   The name of a sector in the **emi_spec.txt** file can be arbitrarily chosen and is followed by the equal sign (‘=’). All sector names chosen in this section form the header of the sector-species matrix (see below).
   For each sector, the data source can be specified as follows:
   - EF_FILE: read emission data from file. This requires filename and variable name (as found in input file) as comma-separated arguments following the EF_FILE keyword. The filename can contain specific tokens which will be replaced by the current date, the model resolution, etc. at runtime (see mo_filename and chapter 3.2 EF_TEMPLATE of this document). Especially %C0 will be replaced by species name as found in the SECTOR-SPECIES MATRIX (see list item below).
   - EF_MODULE: calculate emissions interactively in some other submodule.
   - EF_VALUE: use a globally uniform value (must be given as the next argument)
   - EF_INACTIVE: turn emissions for this sector off

   Another property of the sector source to be defined is its dimensionality. The following values (self-explanatory) are possible: EF_3D, EF_LONLAT, EF_LATLEV, EF_LEV, EF_LAT, EF_SINGLE.

   VERTICAL DISTRIBUTION: For every emission sector the way how emissions shall be distributed vertically has to be defined. Valid tags are **surface** (mass flux will be added to pxtems array as flux boundary condition), **level50m** (this version of boundary condition scheme: mass flux will be added to 2nd lowest model level – very soon next version: level index of 50 m altitude shall be determined at first time step, then emissions will always go into that model level), **volume** (mass flux will be defined as 3D field and added to pxtte, i. e. mass mixing ratio tendency, at all levels) and **fire** (specific handling of vertical mass flux distribution for fire emissions).
2. **SECTOR-SPECIES MATRIX**: The last active section of the **emi_spec.txt** file is the sector-species matrix. It is introduced by the optional mandatory keyword **MATRIX** in the first line of this sector. This matrix provides one line for each species with *nsector* columns. The header line is formed out of the sector names chosen in the sectors section (see above). Each matrix entry is interpreted as a scaling factor (usually 1 to use these emissions or 0 to turn them off). A scaling factor unequal to 1 will only be applied to data that is read in from an external file. If a scaling factor is applied to a species of a sector which is calculated interactively (or even which is constantly set to a specific value by using EF_VALUE), the factor will have no effect on that data.
   You can also write '-' or 'x' to declare that emissions of that species for this sector don't exist. There is a technical distinction between setting an entry to zero or to '-' in that an extra boundary condition will be added in the first case but not in the second case. The results should however be identical. Note that the matrix can contain entries for species not defined in the model. These will be automatically ignored.

3. **Outlook**: The next versions of the handling of the emission matrix will implement further sections in the **emi_spec.txt** file.
   Urgently needed is a section defining species which are behaving in relation to other species, but are no independent emissions by themselves (f. ex. SO4 and NOx). Currently the behavior of these species is hardwired in the code.
   Another proposed extension to avoid the matrix becoming too cluttered with sparse entries, is the introduction of a section named LIST, where sector, species, value triples could be defined one per line.
   There might also arise the need of another section defining diurnal cycle factors or other parameters.

The **emi_spec.txt** file is read via **em_read** (of module mo_emi_matrix); em_read provides the number of sectors defined in the **emi_spec.txt** and the name of the basepath where the emission files are located.
Via **em_get_sector_info** which provides the sectorname and the number of species defined inside this sector given by its index one could loop over all sectors.

Knowing about the number of species inside a sector one can get more information about a special emission by giving the sector index and the species index to **em_get_bc_from_matrix**.

Here is an example **emi_spec.txt** file:

```
# IPCC Emission matrix run T31L39, version 0.92, date: 07.04.2011
# Martin Schultz and Sabine Schroeder, FZ Juelich
# ##############################################################################
# #######    WARNING!! Emissions for MOZART species are incomplete !!     ########
# ##############################################################################
# 1. Sectors
# Notes:
#  a) %C0 will be replaced by species name to identify the correct file
#  b) format is EF_TYPE, parameters
#     for EF_FILE, paramaters are: filename, variable name, options, emission type
#     for EF_MODULE, the only (mandatory) parameter is the emission type
#     for EF_VALUE, a constant value and the emission type must be given
#  c) Valid tags for emission type are:
#     surface  (mass flux will be added to pxtems array)
#     level50m (mass flux will be added to 2nd lowest model level)
#     volume   (mass flux will be defined as 3D field and added to pxtte at all levels)
#     fire     (specific handling of vertical mass flux distribution for fire emissions)
#  d) you can de-activate sectors by preceding them with a comment symbol (#) or by
#     setting EF_TYPE to EF_INACTIVE
DOM=EF_FILE,   %T0/%Y4/emissions_ipcc_%C0_anthropogenic_%Y4_%T0.nc, emiss_dom, EF_LONLAT, surface
ENE=EF_FILE,   %T0/%Y4/emissions_ipcc_%C0_anthropogenic_%Y4_%T0.nc, emiss_ene, EF_LONLAT, surface
IND=EF_FILE,   %T0/%Y4/emissions_ipcc_%C0_anthropogenic_%Y4_%T0.nc, emiss_ind, EF_LONLAT, surface
TRA=EF_FILE,   %T0/%Y4/emissions_ipcc_%C0_anthropogenic_%Y4_%T0.nc, emiss_tra, EF_LONLAT, surface
WST=EF_FILE,   %T0/%Y4/emissions_ipcc_%C0_anthropogenic_%Y4_%T0.nc, emiss_wst, EF_LONLAT, surface
AWB=EF_FILE,   %T0/%Y4/emissions_ipcc_%C0_anthropogenic_%Y4_%T0.nc, emiss_awb, EF_LONLAT, surface
AGR=EF_FILE,   %T0/%Y4/emissions_ipcc_%C0_anthropogenic_%Y4_%T0.nc, emiss_agr, EF_LONLAT, surface
SLV=EF_FILE,   %T0/%Y4/emissions_ipcc_%C0_anthropogenic_%Y4_%T0.nc, emiss_slv, EF_LONLAT, surface
SHIPS=EF_FILE, %T0/%Y4/emissions_ipcc_%C0_ships_%Y4_%T0.nc,         emiss_shp, EF_LONLAT, level50m
AIRC=EF_FILE,  %T0/%Y4/emissions_ipcc_%C0_aircraft_%Y4_%T0.nc,   emiss_air, EF_3D, volume
FFIRE=EF_FILE, %T0/%Y4/emissions_ipcc_%C0_biomassburning_%Y4_%T0.nc, emiss_for, EF_LONLAT, fire
GFIRE=EF_FILE, %T0/%Y4/emissions_ipcc_%C0_biomassburning_%Y4_%T0.nc, emiss_gra,  EF_LONLAT, fire
# FIRE=EF_FILE, %T0/%Y4/gfed2_%Y4%M2%D2.%C0.nc, %SPEC%, EF_LONLAT, biomass_burning
# SOIL=EF_FILE,  %T0/%Y4/HTAP_SURFACE_%C0_%Y4.%T0.nc, soil, EF_LONLAT, EF_IGNOREYEAR, surface
# OCEAN=EF_FILE, %T0/%Y4/HTAP_SURFACE_%C0_%Y4.%T0.nc, ocean, EF_LAT, surface
OCEANI=EF_MODULE, surface
TERRESTRIAL=EF_FILE, %T0/emissions_aerocom_%C0_terrestrial_clim_%T0.nc, DMS_terr, EF_LONLAT, EF_IGNOREYEAR, surface
BIOGENIC=EF_MODULE, surface
DUST=EF_MODULE, surface
SEASALT=EF_MODULE, surface
VOLCC=EF_MODULE, volume
VOLCE=EF_MODULE, volume
VOLCC=EF_INACTIVE
VOLCE=EF_INACTIVE
#
# 2. Species-sector-matrix
# Values in this matrix are interpreted as a scale factor
# Use 0 to switch off emissions usually present
# Use a minus sign (-) to indicate a missing sector for a specific species
# You can enter decimal values here, but the default is to use 1 or zero to make the table look less cluttered
# The MATRIX keyword is mandatory
MATRIX
SPEC    DOM IND TRA ENE WST AWB AGR SLV SHIPS AIRC FFIRE GFIRE SOIL OCEAN OCEANI BIOGENIC DUST SEASALT TERRESTRIAL VOLCC VOLCE
BC      1   1   1   1   1   1   -   -   1     1    1     1     -    -     -      -        -    -       -           -     -
CH4     1   0   1   1   1   1   -   -   1     -    1     1     0    0     -      -        -    -       -           -     -
CO      1   1   1   1   1   1   -   -   1     -    1     1     0    0     -      -        -    -       -           -     -
NO      1   1   1   1   1   1   1   -   1     1    1     1     1    0     -      -        -    -       -           -     -
NO2     -   -   -   -   -   -   -   -   -     -    1     1     -    -     -      -        -    -       -           -     -
OC      1   1   1   1   1   1   -   -   1     -    1     1     -    -     -      -        -    -       -           -     -
SO2     1   1   1   1   1   1   -   -   1     -    1     1     -    -     -      -        -    -       -           -     -
DMS     -   -   -   -   -   -   -   -   -     -    -     -     -    -     1      -        -    -       1           -     -
DU      -   -   -   -   -   -   -   -   -     -    -     -     -    -     -      -        1    -       -           -     -
```

SS	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-
CH3OH	1	1	-	1	1	1	1	1	-	-	1	1	-	-	-	-	-	-	-	-	-
BIGALK	1	1	1	1	1	1	1	-	-	-	1	1	-	-	-	-	-	-	-	-	-
C2H6	1	1	1	1	1	1	1	-	-	-	1	1	-	-	-	-	-	-	-	-	-
C2H4	1	1	1	1	1	1	1	-	-	-	1	1	-	-	-	-	-	-	-	-	-
CH2O	1	1	1	1	1	1	1	-	-	-	1	1	-	-	-	-	-	-	-	-	-
ISOP	-	-	-	-	-	-	-	-	-	-	1	1	-	-	-	-	-	-	-	-	-
C3H8	1	1	1	1	1	1	1	-	-	-	1	1	-	-	-	-	-	-	-	-	-
C3H6	1	1	1	1	1	1	1	-	-	-	1	1	-	-	-	-	-	-	-	-	-
TOLUENE	1	1	1	1	1	1	1	1	-	-	1	1	-	-	-	-	-	-	-	-	-

Annex 1: Rules for netcdf files that can be used as boundary conditions

When the EF_TYPE value is set to EF_FILE, the boundary condition scheme expects to read netcdf files which adhere to the following rules:

- the horizontal resolution must fit the model run's resolution
- all variables must have a "units"-attribute (especially vertical coordinates)
- grid dimensions must be named "time", "lat", "lon", "lev" (or "mlev")
- the time values must be of the type "day(s) since" or "month(s) since" and strictly monotonic increasing, the "calendar"-attribute is always expected to be "standard" (="gregorian") (and is not interpreted by the boundary condition scheme if set to an other value)
- variables must not have additional dimensions as they are supposed to have by the values of EF_GEOMETRY and EF_TIMEDEF (e.g. if a 2D boundary condition is expected, there should be no vertical dimension for this variable)
- filenames must not contain the special character '$'