Memi Technical Report 1

K. Heinke Schlünzen (2006)

Part A - Detailed description of the coding rules for the MeMi model system

A 1. Motivation

The main goals of the coding rules are:

- readability / comprehensibility of the source code
- modular structure with simultaneous independency of the single modules
- uniform module structure
- avoiding semantic mistakes
- improved maintainability
- platform independent code - portability
- uniform error handling
- efficient use of computer resources
- version control
- quality assurance
- quality check

The version control is made with comment lines within the program unit (see section A 5) and for the code units themselves with a tool (Wosik et al., 1992) based on the GNU-rcs (source code management), which uses version numbers for changed units. Thus using the tool older versions are always available and can be re-used. For the model system one single basic version is available for the whole MITRAS/METRAS user group on "niesel" in the directory /pf/u/u232015/rcs/[0]. New or changed modules are put in these libraries after checking of coding rules, code reading and carrying out test runs. The complete external documentation of the model system is managed with the same release system (GNU rcs) and also available for the whole user group (Sub-Directory f90_doc).

A 2. The Fortran 90 standard

Fortran 90 has some more features than FORTRAN 77 and shows some general changes. However, a standard FORTRAN 77 program is still compatible with a Fortran 90 program. For programming new code the new facilities of Fortran 90 should be used. In many cases the use of generic functions produces very good performance, e.g. the use of MATMUL and DOT_PRODUCT.

Some elements from FORTRAN 77 will become obsolete in Fortran 90:

- pause[3]
- entry
- arithmetic if
- loop index with type not equal integer
- common end of several do loops

- do loop end not equal continue or enddo
- branch to endif outside the block if
- alternative return from procedures[4]
- assign and assigned goto statement
- Hollerith constants

These statements are not allowed for use in newly written routines. Also numbered labels are uncommon in Fortran 90. Thus the following changes from FORTRAN 77 to Fortran 90 have to be made:

- goto -> case, if, do while, exit, cycle
- Labelled do -> (Named) do and enddo
- format -> character-parameter
- common-block is written as module. The use of single variables/parameters/functions has to be realised with use <name> ,only : <var>.
- equivalence -> use generic functions.

A.3 Coding Rules

A 3.1 Language

Only the use of Fortran 90 is allowed. The MITRAS/METRAS model system is available for a wide user group, thus all elements and comments have to be written in English.

A 3.2 Elements not being used

- *Functions with side effects are not allowed*. This means that functions containing write or read statements are forbidden. The performance of the code deteriorates, especially if parallelisation is used.
- Within the source code a *series of statements is not allowed (no „;")*. Thus the code is clear and every user can easily read it.
- The *tabulator is not allowed*. In this mannerthe code looks identical in all editors. Within the *emacs the automatic tabulators can be used*, because they are realised as blanks.
- The *attribute dimension is not allowed*
- It is *not allowed to use user defined operators*.User defined data types are only allowed if it is absolutely necessary or if no mathematics is made. (Test runs in December 1996 at the IfT in Leipzig showed that the required time may be up to a factor 30, if these data types are used).

A 3.3 General source code form

- Program units as short as possible. A maximum of *50 executable lines* is recomended.
- At the beginning of each program unit a *comment block* is given (see section A 5). In this comment block all used local variables and parameters are declered in alphabetic order. The global variables and parameters are given in a separate table (see section A 6).

- The *type declaration* has to be made in analogy with section A 5 explicitly, and necessary modules and variables/parameters used in the program unit are to be explained in analogy with section A 5.
- *Comments* should be used in the code to easy the reading of the program. The comment lines have to be marked with a ,,!" in the first column. The first comment line has only the ,,!". Global comments are marked with ,,!" and local comments are marked with ,,!" in the second line. The comment itself starts directly behind one blank following this mark. The comments and therefore this marks have the same incident as the associated source code (default for emacs use), e.g.:

```
PROGRAM test
  INTEGER(KIND=ni) :: iii ,ijj ,ikk
  !
  !... loop indices for calculation of mean values
  DO iii = 1_ni,nx
    !
    !... calculation of mean value for u-component
    umean = ujn(iii)
ENDDO
STOP
end PROGRAM test
```

- All comments have to be in English. The separation of global and local comments may be of interest for a public version of the code. It may be useful to use only global comments because they describe the code completely whereas local comments may be confusing.

- *Blank lines* are not used! Instead empty comment lines have to be used (some compilers do not like blank lines.)
- All Fortran *key words* are written in upcase and all variables, parameter- and method names have to be written in downcase. Within comments uppercase and lowercase letters should be used.

  This is common in many books and codes. The emacs (Version 19.34 or higher) also has standard macros for setting keywords in upcase.

- Each line should be continued after no more than *78 columns*. Then the code on each computer platform can be read with every editor software without line breaks.
- *Indentations:* Below each program unit (program, module, subroutine) the code is indented by 2 columns.

  The code within (DO ,IF) is indented by 3 columns. These are the defaults for emacs use.

- Numbered labels and text output from write statements must start at column 2, because still printers exits which interpret the first letters as printer commands.
- For *continued lines* the last character in the line to be continued has to be the ,,&" and the first character in the intended first column of the following line must be the ,,&". It is not allowed to write any comments behind the ,,&". The code in the following line has to be indented at least 3 columns against the first line. The last character of a line to be continued must not be a ,, , " or a mathematical operator (+ , - , * , /).

  This is useful to reduce errors because a ,,*" at the end of a line to be continued and a ,,*" as the first non blank character in the following line means exponentation and not multiplication!

- *Blanks* have to be used for a readable code.

  Operators (/,*,+,=) have to be surrounded by blanks (<name> * 1.234)
  Parentheses must not be separated with blanks from variables.
  Associated code has to be written in columns, e.g.:
  x = 1.0e0_nr
  ydelta = 1000.0e0_nr
  wert = 0.1e0_nr

- ENDDO,GOTO,ENDIF have to be written in one word without blanks. All other keywords have always to be separated by exact one blank (e.g..: END SUBROUTINE <name>).
- In all program units *IMPLICIT NONE* must be used.
- *Meaningful names* have to be created (see section A 3.4).
- Every *I/O statement* with "IOSTAT=". ,,error=" and ,,end=" must be replaced by "IOSTAT=" (see section A 4).
- *real numbers* must not be checked for identity.
- The declaration of all variables and parametes has to be made in the form: (for classification = is used for space)

  <type> ,<attribute>=::=<name>[,<name>,... ,<name>]

- print is not allowed. For all output the standard output WRITE(<unit>,<format>) must be used. For the report tape (unit 9) it must be used in the form: WRITE (9,...), because to this unit information is written before input like file names is read.

A 3.4 Name conventions

Names must not be identical with keywords.

a) File names

All file names have always the extention .f90. The names, if possible, should not have more than 8 characters (DOS is still alive). If a file name exists in METRAS and in MITRAS and the files contain similar but not identical code (e.g.: oinmet), the file name must use extenions ..._mi.f90, but the method name must be the same (section A 3.4.2).

b) Methods

All names may consist of 11 (+ prefix) at maximum and 2 (+ prefix) characters at minimum.

All methods code starts with "METHODE method name" and ends with "END METHODE method name". The same is valid for do-loops, block if, but the label name is optional.

If the emacs V19.34 or higher is used, it is possible to bind automatically the name behind the end.

All names must only consist of characters, numbers, underscores. The first 3 characters for the method names must be the following:

- program-names: first characters are ,,**pr_**"
- external subroutine-names: first characters are ,,**se_**"
- internal subroutine-names: first characters are ,,**si_**"
- external function-names: first characters are ,,**fe_**"
- internal function-names: first characters are ,,**fi_**"
- module-names: first characters are ,,**mo_**"

  interface-names: first characters are ,,**in_**"

c) Rules for names, variables and parameters

The type and status of variables and parameters is given by the first or the first two characters, respectively (Table A 1). Names for variables and parameters may consist of 11 (+ prefix) and must consist of 4 (inclusive prefix) characters. Loop indices may consist of 3 characters.

The names should only consist characters, numbers and underscores.

Table A 1: Marks for variables and parameters

TYPE	**Variable**		**Parameter**	
	global	**local**	**global**	**local**
POINTER	me	ma	mn	mi
CHARACTER	c *not: cf, cp*	d *not: dp*	cp	dp
INTEGER	n *not: np ,ni ,nr*	i *not: ip*	np	ip
LOGICAL	l *not: lp*	k *not: kp*	lp	kp
REAL	e-h, p-w *not: ep-hp, pp-wp*	a,b *not: ap, bp*	ep-hp, pp-wp	ap,bp
DO-index	-	j	-	-
FORMAT	cf	-	-	
user defined data types[4]	*xg*	*xl*		

A 3.5 Conventions for tape use

The use of the TAPEs is given in the following table:

Table A 2: Tape-Numering

Tape-Nr.	purpose
5	Standard- ***INPUT***
6	Standard- ***OUTPUT***
7	output from pressure solver
8	for control output
9	report tape
10 - 59	additional input files
60 - 99	additional output files
100...	additional temporary test files

A 3.6 KIND-Attribute

Each real/integer variable/parameter must be defined with the kind attribute. The kind attribute is set to **nr** (double precision) for real, **nrs** for real (single precision) and **ni** for integer values.

These variables are declared and initialised within the module mo_kind. For logical and character the default initialisation is used, which is .false. and ‚A', respectively. The advantage of this method is that on all computer platforms the same precision is used. Compiler options like r8 or i64 are thus not needed.

```
MODULE mo_kind
  !
  !------------------------------------------------------------------------
  !     variables for KIND-parameters
  !------------------------------------------------------------------------
  !
  !     CHANGES:
  !     ========
  !     NAME            DATE         COMMEND
  !     ----            ----         -------
  !     Sabine          09/10/96     creation
  !     Heiko           20/04/97     ni, nr added; no parameter declaration
  !     Heiko           11/09/00     including nks for single precison real
  !     Tim Gollnik     31/07/02     ni now specified as nr,
  !     Heinke          20/08/02     nr needs to be kind 8, ni kind 4
  !                                  when compiled with single precision,
  !                                  otherwise the default functions are
not
  !                                  on all machines known and libraries
  !                                  (e.g. plotting) might not read binary
```

```
data
  !     Heinke          09/07/03     add save
  !     Heinke          03/01/06     delete obsolete commented lines
  !
  !     METHOD:
  !     =======
  !
  !     LITERATURE
  !     ==========
  !     ---NO---
  !
  !-------------------------------------------------------------------------
  !
  !     USE BY:                      TYP
  !     =======                      ---
  !                                  MODULE
  !
  !     FILES:
  !     ======
  !     NAME                         ATTACH
  !     ----                         ------
  !
  !     SUBROUTINES:
  !     ============
  !     NAME
  !     ----
  !
  !-------------------------------------------------------------------------
  !
  !     INPUTPARAMETER:
  !     ===================
  !     NAME              TYPE       COMMENT
  !     ----              ----       -------
  !     ---NONE---
  !
  !     OUTPUTPARAMETER:
  !     ================
  !     NAME              TYPE       COMMENT
  !     ----              ----       -------
  !     ---NONE---
  !-------------------------------------------------------------------------
  !
  !
  !---  TYPENANWEISUNG  MODUL /XKIND/:
  !
  !
  !... Definition of precision parameters
  !
  INTEGER, PARAMETER :: nr = 8
  !
  !... for single precision output:
  INTEGER ,PARAMETER :: nks = 4
  !
  !... for integer kind value is used:
  !
  INTEGER, PARAMETER :: ni = 4
  !
  !
  SAVE
  !
END MODULE mo_kind
```

A 3.7 INTENT - attribute

The intent attribute is specifying the way of use of formal parameters. This attribute must be used for subroutine or function declarations if formal parameters are present.

If the intent attribute is given for a variable, additional data, parameter, pointer or save attributes must not be used.

A 3.8 Modules

The modules contain the declaration of global variables and parameters. The initialisation and allocation of space for those variables/parameters is made in separate initialisation routines.

Only the modules mo_functions and mo_exception contain executable program code.

The module mo_functions contains all external functions for the model system. The source code of the functions is located outside the module in single files and is then bound into the module via include. This facilities the use in other programs like pre- and post-processors.

In the file names_object_f90 on /pf/u/u232015/f90_doc the modules listed are with this contents.

A 3.9 Exception handling

Error handling should be similar in all parts of the model. The program unit se_errmsg.f90 treats "informative", "warning" and "error"-messages. They are similar in their structure and are called everywhere in the model system of MITRAS/METRAS in the same way.

A 3.10 Dynamic memory allocation

In Fortran 90 a dynamical memory allocation is used. The advantage is that one executable code can be used for every domain up to the hardware limits. One problem is that memory management must be done by the programmer. If allocated memory is not deallocated explicitly, too much memory is used and the performance will be worse.

For an optimal and transparent use of memory the user must:

- dimension arrays explicitly (e.g.: real **a_feld(nx3,nx2,nx1)**)
- If allocate is used, the memory must be explicitly cleared by deallocate.
- The user must test the performance of allocate and deallocate. For this option the (de)allocate statements are available.

It must be tested if the memory is really deallocated in case of repeated memory allocation (e.g. in a subroutine).

A 4. Example program

You can find this file under /pf/u/u232015/rcs/f90_doc on *niesel* with the file name pr_example.f90.

```
PROGRAM pr_example
  !
  !-------------------------------------------------------------
  !--- This is an example for a program following the programming
  !--- guidline
  !-------------------------------------------------------------
  !
  !
  ! Changes:
  ! ========
  ! Name              Date          Comment
  ! ----              ----          -------
  ! Heiko             09/07/97      typing of example program
  !
  !-------------------------------------------------------------
  ! Method:
  ! =======
  ! In this program it is only intended to show the general
  ! program layout
  !
  !-------------------------------------------------------------
  !
  ! Datafiles:
  ! ==========
  ! Name:             Attach:
  ! -----             -------
  ! TAPE6                W
  !
  ! Subroutines
  ! ===========
  ! se_intern1
  !
  ! FUNCTIONS:
  ! ==========
  !
  ! MODULE:                  USE:
  ! =======                  ====
  ! mo_kind                  ni ,nr
  !
  ! Parameter for Subroutine use:
  ! =============================
  ! Name              Type                    Comment
  ! ----              ----                    -------
  !
```

```
  !
  !-------------------------------------------------------------
  !
  USE mo_kind ,ONLY : ni ,nr
  IMPLICIT NONE
  !
  INTEGER(KIND=ni) ,PARAMETER :: iparam1=25 ,iparam2=2 ,iparam3=12
  REAL(KIND=nr) :: windu, windv
  !
  DO jk=1,iparam1
     windu = windv * 2.75e0
  ENDDO
  !
  WRITE(6,*) ' M3TRAS ended'
  !
  STOP
END PROGRAM pr_example
!
CONTAINS
  !
SUBROUTINE si_intern1
  !
  !-------------------------------------------------------------
  !--- This is an example for a program following the programming
  !--- guidline
  !-------------------------------------------------------------
  !
  !
  ! Changes:
  ! ========
  ! Name              Date          Comment
  ! ----              ----          -------
  ! Heiko             09/12/91      typing of example program
  !
  !-------------------------------------------------------------
  ! Method:
  ! =======
  ! In this subroutine it is only intended to show the general
  ! program layout
  !
  !-------------------------------------------------------------
  !
  ! Datafiles:
  ! ==========
  ! Name:             Attach:
  ! -----             -------
  !
  !
  ! Subroutines
  ! ===========
  !
  !
  ! FUNCTIONS:
  ! ==========
  !
  ! MODULE:                  USE:
  ! =======                  ====
  ! mo_kind                  ni ,nr
  ! mo_xpara                 nx1 ,nx2 ,nx3
  !
  ! Parameter for Subroutine use:
  ! =============================
  ! Name              Type                    Comment
```

```
  ! ----              ----                    -------
  !
  !
  !-----------------------------------------------------------
  !
  USE mo_kind ,ONLY : ni ,nr
  USE mo_xpara ,ONLY : nx1 ,nx2 ,nx3
  !
  IMPLICIT NONE
  !
  INTEGER(KIND=ni) :: iii ,ijj ,ikk
  !
  DO ikk = 1,nx3
     DO ijj = 1,nx2
        DO iii = 1,nx1
           u0(ikk,ijj,iii) = 12.03e0 + ujn(ikk,ijj,iii-3)
        ENDDO
     ENDDO
  ENDDO
  !
  RETURN
END SUBROUTINE si_intern1
```

A 5. References

[1] B. und R. Wojcieszynski, 1993:
Fortran 90; Programmieren mit dem neuen Standard: Addison Wesley

[2] D. Rabenstein, 1995:
Fortran 90, Lehrbuch: Hanser

[3] Fortran90 -Ein Nachschlagewerk, 1993
RRZN (Regionales Rechenzentrum für Niedersachsen/Universität Hannover, 2. Auflage

[4] Europäische Wetterdienste, 1996:
Europeanstandards for writing and documenting exchangeable Fortran 90 code,Version 1.1
[5] Wosik, J., H. Schlünzen, K. Bigalke, 1992

PROTOOL: EinSource-Code-Verwaltungs-System auf Basis von RCS unter UNIX;
InternerBericht, Meteorologisches Institut, Universität Hamburg

---

[0] Libnames are changed for large updates. A list of available libraries is:
/pf/u/u232015/res/info_versions
[1] lib can be: 1d, 3d, gen, emi.
[2] Does not exist in METRAS and is not allowed for batch use.
[3] The calling program is continued with a statement order different than the normal corresponding statement.
[4] This data type is only allowed to be used if the compiler can optimise it or if no mathematics is done.

Part B - Coding rules - short form for MeMi model system

B 1. General rules:

programming language	Standard Fortran 90
line length	max. 78 columns
Extension of files	.f90 or _mi.f90 (if seperate routine exists for MITRAS)
program units	if possible below 50 lines of executable code
comments	starting with,,!"; indent like associated code
	first line only ,,!"
	global comments in second and following lines marked with ,,!--- "
	local comments in second and following lines marked with ,,!... "
	all comments indented as associated code (default in emacs)
	all comments in English
blanc lines	not allowed ==> blank comment lines(,,!")
for all units	**IMPLICIT NONE**
KIND-attribute	**LOGICAL** and **CHARACTER**: default values **INTEGER**-variables and parameter **KIND** =**ni** **REAL**-Variables and parameter **KIND** =**nr**
REAL constants	always in exponential form: 1015.00e0_nr
DIMENSION-attribute	not allowed
names	self describing names
Fortran keywords	upcase
variables and parameter	downcase
MODULE	**USE** <modulename> , **ONLY** : <var>[<,var>< ,var>]
more than one statement per line	not allowed (no ,, ; ")
code block (**DO**, **DO WHILE**, **block IF**, **CASE**, **INTERFACE** etc.)	indent: 3 columns
continuation lines	last character of line to be continued must be ,,&" indent of code: 3 columns at minimum first character of continuation line must be ,,&" in intended column, blank thereafter
	,, , " or mathematical operator (+,-,*,/) are not allowed as the last character of line to be continued
Blanks	use them! operators must be surrounded by exactly one blank

	in front of a parenthesis blanks must not be used blank behind comma is not allowed
READ	always with „**IOSTAT**=" and handling of exceptions
END	**ENDDO**, **ENDIF**, **GOTO** must be written together (no blanks) others: **END** keyword (e.g..: **END SUBROUTINE**)
Arrays	dimension always explicit (e.g..: **REAL** (**KIND=nr) a_feld(nx3,nx2,nx1)**)
functions with side effects	not allowed
REAL values	must not be checked for identity(not allowed: **EQ**)
PRINT	must not be used; use instead: **WRITE** (6,*)
declarations	<Type> ,<Attribute> :: <name>[,<name>... ,<name>]
user defined data types	use only if optimization is possible or no mathematics is done!
INTERFACE	all necessary **INTERFACE**s put in a few **MODULE** s
INTENT	use for all keyword parameters

B 2. Conventions for names

- If in METRAS and MITRAS the same routines exist with different code, the MITRAS code must have the extension _mi.f90 for the file names but must not have any more differences in the name.
- names must not be the same as keywords.
- all methods start with "<methode> method name" and end with "**END** <methode> method name". The same is valid for **DO**-Loops, Block **IF** etc., but the label is optional.
- names for methods may consist of a maximum of 11 (+ prefix) and a minimum of 2 (+ prefix) characters.
- The type and status of the variable or parameter is defined by the prefix.
- names for variables and parameters may consist of 11 (incl. prefix) and must consist of at least 4 (incl. prefix) characters. Loop variables may only have 3 characters.<!--StartFragment -->

Table B 1: First character for variables and parameters.

TYPE	**Variable**		**Parameter**	
	global	**local**	**global**	**local**
POINTER	me	ma	mn	mi
CHARACTER	c *not: cf, cp*	d *not: dp*	cp	dp
INTEGER	n *not: np ,ni ,nr*	i *not: ip*	np	ip

LOGICAL	l *not: lp*	k *not: kp*	lp	kp
REAL	e-h, p-w *not: ep-hp, pp-wp*	a,b *not: ap, bp*	ep-hp, pp-wp	ap,bp
DO-index	-	j	-	-
FORMAT	cf	-	-	
user defined data types[4]	*xg*	*xl*		

Table B 2: Prefix for methods.

Methods	**prefix**
PROGRAM	*pr_*
external **SUBROUTINE**	*se_*
internal **SUBROUTINE**	*si_*
external **FUNCTION**	*fe_*
internal **FUNCTION**	*fi_* (old : o)
MODULE	*mo_*
INTERFACE	*in_*

B 3. Rules for the labels

Numeric labels have been disallowed, instead:

Statement	**Label/ Structure without Label**
STOP (error)	use **IOSTAT** and exception handling
ERROR (read)	
END (read)	
CASE, **IF**, **DO WHILE**, **EXIT**, **CYCLE**	names for labels
Labelled **DO**	names for labels
FORMAT	**CHARACTER**-Parameter

B 4. Conventions for tape use

Tape-Nr.	**purpose**

5	Standard- *INPUT*
6	Standard- *OUTPUT*
7	output from pressure solver
8	for control output
9	report tape
10 - 59	additional input files
60 - 99	additional output files
100...	additional temporary test files

Part C - Documentation rules for the MeMi model system

C 1. Preface

Only few proposals were made concerning the documentation of meteorological computer programs. In 1994 the Model Evaluation Group (MEG) of the European Communities released ,,guidelines for model developers" [1] and the ,,model evaluation protocol" [2] . The guideline on "Evaluation for flow around buildings and obstacles" for prognostic microscale wind field models [3] includes some recommendations. A more general guideline for documentation of software products was published in 1980 by the German DIN Institute [4] in form of DIN 66230-1980.

Following the presented specifications, a main requirement of a model documentation is its completeness. The documentation must be up to date and readable for each user. The documentation of software products and thus for numerical models may be split into two categories: The external documentation outside the code and the internal documentation inside the code.

C 2. General specifications

For the MeMi model ystem the documentation is divided into external and internal documentation. The whole documentation has to be written in English. The external model documentation is consistant with VDI 3783-9 [3].

The following sections specify the documentation structure in detail . The important points to be regarded by the source code developer are mentioned.

C 2.1 External Documentation

For practical use the external documentation is split into three parts: *Brief Description*, D*etailed Documentation* and *User Manual*. The main topics of the external model documentation package are the description of the model and description of data sets used for the development of the code (e.g. for numerical constants). A scientific evaluation of the model has also to be given.

a) Brief description

This description is different for every model of the MeMi model

System (METRAS,MITRAS,MECTM,MICTM)

1. model name, version number, date of present model publication.
2. publisher, contact person
3. field of application
4. limits of application

5. area size, model resolution
6. permissible expansions of grid width
7. solution patterns
8. input parameters, output parameters
9. validationion and evaluation
10. hardware and software requirements of the programme
11. availability
12. group of users
13. bibliography

b.) Detailed Description

The *detailed description* of the model should answer detailed questions about the model and the used programming code. The task, basic equations, approximations, parameterisations as well as the boundary conditions have to be discussed in detail. The data used for calibrations have to be mentioned, and the quality of the used data sets for calibration has to be evaluated. All units used within the model, especially within chemical reaction parts, input data sets and output datasets, have to be listed. The solution technique and initialisation shall be described in detail. Instructions on how to interpolate in detail. Instrucions on how to interpolate the model results into grid points which have not been used in the model shall be provided. The validation and evaluation of the model with an explanation of used data sets have to be fully discribed. This description is different for every model of the MeMi model System (METRAS, MITRAS, MECTM, MICTM).

The table of content of this detailed description is:

1. Task
2. Basic equations
3. Approximations
4. Parameterisations
5. Boundary conditions
6. Numerical treatment
7. Nesting
8. Initialisation
9. Data sets used for calibration of the model
10. Units
11. Validation and evaluation
12. Limitations
13. References

c.) User Manual

The *user manual* , is developed to give the user all necessary information to get easily started. This documentation jans all MEMI modells. It shall ensure that same meaning are troughout all models of the MEMI model system.

Its content is:

1. Outline of MEMI model system
2. a.) Installation procedure b.) User interface
3. Hard- and software requirements
4. Interruption and restart features
5. Used programming language (standards)
6. Error messages
7. Data input
8. Implemented boundary conditions
9. Names of species and reaction systems
10. Table of global variables
11. Table of constants including their value and unit
12. Data - flow diagram
13. Funktional diagram (calling tree)
14. Data output
15. Application example

The global variables have to be explained. The structure of the ASCII file is given in Table A 3. You can find the associated TeX-file under /pf/u/u232015/rcs/f90_doc on *niesel* at DKRZ with the file name names_varpar_f90.tex.

<>

Table C 1: Structure of the ASCII-file for documentation of global variables

variable	defined in module	symbol in model description	short explanation

What has the developer to do for external documentation ?

The developer of new or changed code has to make an external description of the new method used, the input and output values (structure numbers and variable names) and documentation of validation runs. References have to be given for the description of the used method. For validation detailed information about the target value, initialisation data sets, data sets for comparisons, analytical solutions, processes studied and references have to be given, too. The result should state if the data set, analytic solution, or sensitivity parameters are useful for a model validation with respect to the chosen strategy.

In addition, for the *detailed description a* description of new equations, new approximations, parameterisations, boundary conditions, numerical treatment etc. have to be given.

For the third part of external documentation the user interface, data input and output with information on units, error messages, global variables and calling tree must be given. Please note that the documentation has to be written in TEX and priviede in digital form.

C 2.2 Internal documentation

Documentation must be available also (see Section A4) within the code. The header of each program unit gives information about the history of development (different users who made changes and the kind of changes). Also the method used should be described shortly and references have to be given. Global comments (!---) within the executable code have to explain the code for external users wheras local comments (!...) should address programming issues.

What has the developer to do for internal documentation ?

Use comments ! Put the header given in section A 4 in each program unit and describe the specific method used and the task of the unit as well as the necessary data files, other program units and local variables.

Explain the structure of the code to each user with global, and in addition local comments and give the units used in your equations. If complex Fortran structures are used, please make them clear with local comments.

C 3. References

[1] Guidelines for model developers; Model Evaluation Group; 1994
European communities, Directorate-general XII, science research and development

[2] Model evaluation protocol; Model Evaluation Group; 1994
European communities, Directorate-general XII, science research and development

[3] Verein Deutscher Ingenieure (2005): Environmental Meteorology: Prognostic microscale eind field model-Evaluation for flow around buildings and obstacles VDI 3738, Part 9; Komission Reinhaltung der Luft im VDI und DIN, Beuth Verlag GmbH, Berlin, Germany

[4] DIN 66230 - 1980[4] A.Ebel, E. Schaller, K.K. Schlünzen; 1997:
Evaluierungsstrategie für Chemie-Transport-Modell-Systeme im TFS-LT1; in Vorbereitung