Max-Planck-Institut
für Meteorologie
Bundesstr. 55
D-20146 Hamburg

Deutscher Wetterdienst
Kaiserleistr. 42
D-63067 Offenbach

Programming Guide for ICON

in alphabetical order:
Luca Bonaventura*, Monika Esch*, Helmut Frank**,
Marco Giorgetta*, Thomas Heinze**, Peter Korn*,
Luis Kornblueh*, Detlev Majewski**, Andreas Rhodin**,
Pilar Rípodas**, Bodo Ritter**, Uwe Schulzweida*, ...

August 10, 2005

*Max-Planck-Institut für Meteorologie, Bundesstr. 55, D-20146 Hamburg, Germany
**Deutscher Wetterdienst, Kaiserleistr. 42, D-63067 Offenbach, Germany

Contents

Part I.
Programming Conventions

1. Introduction

The programming standard on which the following conventions are based mostly is the *European Standards For Writing and Documenting Exchangeable Fortran 90 Code*, written by Andrews, Phillip (UKMO), Gerard Cats (KNMI/HIRLAM), David Dent (ECMWF), Michael Gertz (DWD), and Jean Louis Ricard (Meteo France). We mainly refer to their paragraph "Coding rules for routines".

ICON, as a new development, is expected to follow the most recent version of the Fortran standard, which is Fortran 95. After its official release it will be Fortran 2003. The first section of this document contains a brief explanation of the more important language elements that are used in ICON. Problems with specific compilers on various platforms are indicated where known. The model has been successfully compiled with the following compilers by date of print:

- NAGWare Fortran 95 compiler Release 4.2(517), Linux x86
- NAGWare Fortran 95 compiler Release 5.0(253), Linux x86
- Lahey/Fujitsu Fortran 95 Compiler Release L6.20a, Linux x86

The following compiler does not properly work:

- unknown

If you find bugs or fixes for any of these compilers, please notify the authors. Also, if you would like to contribute to this document, please let us know!

The second part of this document describes and explains the ICON programming rules and conventions in detail. A reference with brief examples is given to facilitate the development of new standardized code.

The programming guidelines deal with several aspects of the programming process and should serve the following goals:

- writing of readable/comprehensive source code
- modularizing of code using more or less independent `MODULE`s
- standardizing the look and usage of `MODULE`s
- avoiding semantical errors
- reducing the maintenance cost
- creating machine independent source code
- well defined exception/error handling
- version control
- quality management and control

As introduction into the new language features provided by Fortran 95 the following section introduces the important features for the readers convenience.

2. The Fortran 95 standard itself - a short introduction and remarks

With a few minor exceptions, Fortran 95 is a superset of X3.9-1978 FORTRAN (FORTRAN77). Yet, FORTRAN77 code will usually not be portable without changes. For example, many (if not most) programmers employed constructs beyond the FORTRAN77 standard, or relied on the behavior of specific compilers.

What are the new features of Fortran 95?

A complete description of the differences between FORTRAN77 and Fortran 95 clearly goes beyond the purpose of this document. The interested reader may consult a good book (e.g. Fortran 95 explained by Metcalf & Reid, Oxford Science Publications, New York, 1996) or she may find information on one of the following Internet sites:

- Fortran 90 for the FORTRAN77 programmer (tutorial):
  `http://www.nsc.liu.se/~boein/f77to90/f77to90.html`
- SciCon Fortran 90 tutorial at NASA Ames:
  `http://www.nas.nasa.gov/Groups/SciCon/Tutorials/FORTRAN90/`
- Fortran 90 site at King's college:
  `http://www.kcl.ac.uk/kis/support/cc/fortran/f90home.html#1.0`
- High Performance Object-Oriented Programming in Fortran 90/95:
  `http://www.cs.rpi.edu/~szymansk/oof90.html`

Some of the most important changes from FORTRAN77 to Fortran 95 are:

- array notation (operators, etc.)
- dynamic memory allocation
- derived types and operator overloading
- keyword argument passing, `INTENT (in, out, inout)`
- modules, including the attributes `PRIVATE` and `PUBLIC`
- modern control structures
- free format source code form
- more intrinsic functions/subroutines

Fortran 95 allows the FORTRAN77 programmer to write code faster, to make it more legible, and to avoid typical, sometimes hard to find errors. And finally is in line with scientific and industrial engineering communities where Fortran is and is going to remain for a good while the favorite language.

Fortran 95 is a minor revision of Fortran 90. The 3 main extensions with respect to that previous version are:

- `FORALL` statement and construct
- `PURE` and `ELEMENTAL` procedures
- Structure and pointer default initialization

3. The ICON programming standard

3.1. Programming languages

All model components should be written in standard Fortran 95 and not rely on any specific vendor extension. There is enough difficulty involved in porting standard conforming code from machine to another. Please don't make life even more difficult trough bypassing the standard.

Owing to the international user community, all naming of variables, modules, functions, and subroutines as well as all comments are to be written in English. In case you encounter a situation which you cannot solve in Fortran 95 (e.g. you need to include a library written in C), please make sure the non Fortran code only uses ANSI C with the POSIX extensions. This type of code should go into the support library.

3.2. Obsolete features and replacements for this

The following summary lists features that are deprecated or obsolete in Fortran 95. It also contains some typical FORTRAN77 *tricks* which make it hard to maintain code and port it to other platforms. The intercomparison of FORTRAN77 and Fortran 95 features given here, shall, provide a first set of programming rules.

- `COMMON` blocks - use the declaration part of `MODULE`s instead.
- `EQUIVALENCE` - use `POINTER`s or derived data types instead to form data structures.
- Assigned and computed `GOTO`s - use the `CASE` construct instead.
- Arithmetic `IF` statements - use the block `IF`, `ELSE`, `ELSE IF`, `END IF` construct instead.
- Labels
  - Labeled `DO` constructs - use unlabeled `END DO` instead[1].
  - I/O routine's `END` and `ERR` use `IOSTAT` instead.
  - `FORMAT` statements: use character parameters or explicit format specifiers inside the `READ` or `WRITE` statement instead. In general a message subroutine is provided for informations which should be used, because it supplies a single point of IO unit specifications.
  - Avoid any unused statement like a labeled `CONTINUE` not being jumped to.
- `GOTO`
  - The only sensible use of `GOTO` is to jump to the error handling section at the end of a routine on detection of an error. The target label should be a `CONTINUE` statement the label should be 999. However, it is recommended to avoid this practice and use `IF`, `CASE`, `DO WHILE`, `EXIT` or `CYCLE` statements or a contained `SUBROUTINE` instead. If you feel you cannot avoid a `GOTO`, then add a clear comment to explain what is going on and why you need to use `GOTO`. Also add a comment to the labeled `CONTINUE` statement.
- `PAUSE` - see remarks about `STOP` on page 11
- `ENTRY` statements: a subprogram may only have one entry point.
- Fixed source form

[1] Due to the sometimes complicated structure of the physics routines and the large amount of scientists working on them, we recommend for the ease of communication labeled do constructs of the `DO` - labeled `END DO` construct.

- Avoid functions with side effects, i.e. functions that alter variables in their argument list or in modules used by the function, or functions which perform I/O operations. Although this is common practice in C, there are good reasons to avoid side effects. First, the code is easier to understand, if you can rely on the rule that functions don't change their arguments, second, some compilers generate more efficient code for *pure* (in Fortran 95 there are the attributes `PURE` and `ELEMENTAL`) functions because they can store the arguments in different places. This may become especially important on massive parallel machines.

- Do not implicitly change the shape of an array when passing it into a subroutine. Although actually forbidden in the FORTRAN77 standard it was very common practice to pass n dimensional arrays into a subroutine where they would, say, be treated as a 1 dimensional array. Though officially banned in Fortran 90, this practice is still possible with external routines for which no interface block is supplied. The danger of this method is that it makes certain assumptions about how the data is stored. Note: we would like to encourage the use of BLAS and LAPACK which implies the before mentioned array collapsing. This is acceptable in this case, because the performance improvement is worth the 'dirty tricks' and well marked by the BLAS and LAPACK usage.

- Try to avoid to use `DATA` and `BLOCK DATA`. This functionality is given by initializers in Fortran 95.

4. Style rules or how should code look like

The general objective behind a style guide is to write portable code that is easily readable and can be maintained by different people. Many rules follow common sense and should be obvious. We note, that many of the formatting suggestions are easily achieved if you use the GNU emacs (or xemacs) editor in Fortran 90 mode. Although it is always tempting for scientists to restrict coding efforts to their specific application, we strongly urge you to try and write the code in a more general form. This will make it much easier to make modifications or couple your routine to others. As an example, you should never rely on a specific model resolution (horizontally or vertically). It may take somewhat longer to get your code to run in the first place, but think of the fact that properly written code can drastically reduce maintenance costs and helps to prevent errors.

Here are the most important rules that we would ask you to adhere to:

- Use free format syntax.

- For viewing on terminals or printing 78 characters per line are convenient. But Fortran 90 allows a line length of up to 132 characters.

- Any date follows the ISO 8601 standard. That is: YYYY-MM-DD HH:MM:SS.

- Use `IMPLICIT NONE` in all program units. This ensures that all variables must be explicitly declared, and hence are documented. It also allows the compiler to detect typographical errors in variable names. For `MODULE`s, one `IMPLICIT NONE` statement in the modules definition section is sufficient.

- Use meaningful English variable names. The names of global variables and constants are given in section F and G of the appendix (see also the remarks in section 5).
  Within a subroutine or function we adopt the naming conventions given by the DOCTOR standard as described in *ECHAM3 - Atmospheric General Circulation Model, 1992* with some modifications described as follows.

  Variable names begin with significant prefix letters. These prefix letters not only indicate the type of variable (`REAL`, `INTEGER`, etc.), regardless the size, but also the status of the variable (local, argument, module contained, etc.).

– Variable type

`INTEGER` variables should follow the FORTRAN77 standard and be prefixed by `i`, `j`, `k`, `m` or `n`. `LOGICAL` variables begin with `l`. `CHARACTER` variables should have meaningful names and are defined according the required length (this is different to the previous convention which was fixed to `*8`). All other variables are of type `REAL`.

But note that all variables have to be declared anyway.

– Variable status

This is intended to be used with passed dummy arguments:

type	prefix
`INTEGER`	`k_`
`REAL`	`p_`
`LOGICAL`	`ld_`

Local variables:

type	prefix
`INTEGER`	`i`
`INTEGER` (loop control)	`j`
`REAL`	`z_`
`LOGICAL`	`ll_`

Pointers:

type	prefix
`POINTER`	`ptr_`

Some additional rules are required for new structures as are:

Names for modules also containing data should start with `mo_`; names for modules only containing subroutines should start with `m_`.

Abbreviations are acceptable as a means of preventing variable names getting too long, but still be descriptive.

- Fortran keywords should be written in upper case, the remaining code in lower case.
- To improve readability indent code within `DO`; `DO WHILE`; block `IF`; `CASE`; `INTERFACE`; etc. constructs by 2 characters.
- Similarly, try to make equations recognizable and readable as equations. Readability is greatly enhanced by starting a continuation line with an operator placed in an appropriate column rather than ending the continued line with an operator.
- Where they occur on separate lines indent comments to reflect the structure of the code.
- Use blank space, in the horizontal and vertical, to improve readability. In particular try to align related code into columns. For example, instead of:

```
! Initialize Variables
i=1
z_meaningfulname=3.0_wp
z_SillyName=2.0_wp
```

write:

```
! Initialize variables
i                = 1
z_meaningfulname = 3.0_wp
z_silly_name     = 2.0_wp
```

- Do not use tab characters in your code: this will ensure that the code looks as intended when ported.
- Separate the information to be output from the formatting information on how to output it on I/O statements. E.g. don't put text inside the brackets of the I/O statement.
- All variables and routines must be encapsulated in a `MODULE` and the `PRIVATE` attribute should always be introduced immediately after the `IMPLICIT NONE` statement. What has to be used outside the module must be explicitly declared `PUBLIC` instead.
- Use `USE` with the `ONLY` attribute to specify which of the variables, type definitions etc. defined in a module are to be made available to the using routine. Of course you don't need to add the `ONLY` attribute if you include the complete module or almost all of its public declarations.
- Variables used as constants should be declared with attribute `PARAMETER` and used always without copying to local variables. This prevents from using different values for the same constant.
- Pack all standalone subroutines in a module called m_name of the subroutine to allow for argument checking.

4.1. Automatic documentation

Each function, subroutine, or module will include a prologue instrumented for use with the ProTEX auto-documentation script (http://dao.gsfc.nasa.gov/software/protex). The purpose is to describe what the code does, possibly referring to external documentation. The prologue formats for functions and subroutines, modules, and header files are presented in the appendix. In addition to the keywords in these templates, ProTEX also recognizes the following:

```
!BUGS:
!SEE ALSO:
!SYSTEM ROUTINES:
!FILES USED:
!REMARKS:
!TO DO:
!CALLING SEQUENCE:
!CALLED FROM:
```

These keywords may be used at the developer's discretion. We would like to recommend to use the `REMARKS` part to add an responsible author, who can be contacted in case of problems.

Within the ICON project an extension of the original perl script **protex** called **iconprotex** should be used. It includes ICON specific key words and will be provided on the ICON web page.

4.2. Interface blocks

Explicit interface blocks are required between f95 routines if optional or keyword arguments are to be used. They also allow the compiler to check that the type, shape and number of arguments specified in the `CALL` are the same as those specified in the subprogram itself. In addition some compilers (e.g. the Cray f90 compiler) use the presence of an interface block in order to determine if the subprogram being called is written in f95 (this alters how array information is passed to the subroutine). Thus, in general it is desirable to provide explicit interface blocks between f95 routines. There are several ways to do this, each of which has implications for program design; code management; and even configuration control. The one we select is based on automatic interface blocks.

Fortran 90 compilers can automatically provide explicit interface blocks between routines following a `CONTAINS` statement. The interface blocks are also supplied to any routine using the containing module. Thus, it is possible to design a system where no interface blocks are actually coded and

yet explicit interface blocks are provided between all routines by the compiler. One way to do this would be to modularize the code at the f95 module level, i.e. to place related code together in one module after the `CONTAINS` statement. Routine a, in module A calling routine b in module B would then only have to use module B to be automatically provided with an explicit interface to routine b. Obviously if routine b was in module a instead then no use would be required. One consequence of this approach is that a module and all the routines contained within make up a single compilation unit. This may be a disadvantage if modules are large or if each module in a package contains routines which use many other modules within the package (in which case changing one routine in one module would necessitate the recompilation of virtually the entire package). On the other hand the number of compilation units is greatly reduced, simplifying the compilation and configuration control systems.

The greatest advantage of this approach is that the compiler does the work of providing the interface blocks, thereby reducing programming overheads, and at the same time guaranteeing that the interface blocks used are correct.

Inside of the models physics packages we would like to have the physical parameters passed as arguments. This follows a proposal by Kalnay and potentialy allows for simpler exchange with foreign packages.

4.3. Array notation

Array notation should be used whenever possible. This should help optimization regardless what machine architecture is used (at least in theory) and will reduce the number of lines of code required. To improve readability the array's shape should be shown in brackets, e.g.:

```
onedarraya(:)   = onedarrayb(:) + onedarrayc(:)

twodarray(:, :) = scalar * anothertwodarray(:, :)
```

When accessing sections of arrays, for example in finite difference equations, do so by using the triplet notation on the full array, e.g.:

```
twodarray(:,2:len2) = scalar                          &
&                   * (anothertwodarray(:,1:len2-1) &
&                   -  anothertwodarray(:,2:len2))
```

4.4. More styles

Here more recommendations (or general rules) on the usage of certain programming conventions are given. We expect this to be useful for a better understanding of the code.

- Each single entity of a source code is at least one of the routine types. Always name program units and always use the `END PROGRAM`; `END SUBROUTINE`; `END INTERFACE`; `END MODULE`; etc constructs, again specifying the name of the program unit. This helps finding the end of the current program entity. `RETURN` is obsolete and so not necessary at the end of program units.
- Use of `>`, `>=`, `==`, `<`, `<=`, `/=` instead of `.GT.`, `.GE.`, `.EQ.`, `.LT.`, `.LE.`, `.NE.` in logical comparisons is recommended. The new syntax, being closer to standard mathematical notation, should be clearer.
- Usage of the `DIMENSION` statement or attribute is not necessary. Declare the shape and size of arrays inside parentheses after the variable name on the declaration statement.
- The `::` notation is quite useful to show that this program unit declaration part is written in f95 syntax, even if there are no attributes to clarify the declaration section.

- Declare the length of a character variable using the `CHARACTER (len=xxx)` syntax.
- We recommend against the use of recursive routines on efficiency grounds for computational intensive routines (they tend to be inefficient in their use of CPU and memory).
- To improve portability between 32 and 64 bit platforms, it is necessary to make use of kinds as provided by `mo_kind` to obtain the required numerical precision and range as well as size of `INTEGER`. A module is already provided (mo_kind.f90) and attached to this guide as section E of the appendix. It should be noted that constants need to have attached a _kindvalue to have the according size.

  Example:

```
USE mo_kind, ONLY: wp

IMPLICIT NONE

! Declaration of a constant

REAL(wp), PARAMETER :: a_hour = 3600._wp  ! 1 hour in seconds

! Declaration of a 2d field

REAL(wp), POINTER   :: z_snowcover(:,:)

! Declaration of a local variable

REAL(wp) :: z

              ⋮

z = 4.0_wp
```

  Provided are the following kinds in `mo_kind`:

kind variable	assumed number of bits
real variables	
`sp`	32
`dp`	64
integer variables	
`i4`	32
`i8`	64

  Note:
  The bit sizes given are not mandatory. The kind value is selected regarding a given precision, which results on current systems in this bit sizes. This may change in future.

  We recommend to use the working precission `wp` for all reals.`wp` is set in `mo_kind`. With the advantage porting the code to another platform the changing of just one line is necessary.

- A rule which is quite obvious is to use always generic functions instead of the obsolescent specific intrinsic functions.
- Never use a Fortran95 keyword as a name of routines or variables!

- When an error condition occurs inside a package, a message describing what went wrong will be printed. The name of the routine in which the error occurred must be included. It is acceptable to terminate execution within a package, but the developer may instead wish to return an error flag through the argument list. If the user wishes to terminate execution within the package, a generic ICON coupler termination routine `finish` must be called instead of issuing a Fortran STOP. Otherwise a message-passing version of the model could hang.

5. Global variable name definitions

The important variables with global scope are supposed to follow the CF-Conventions as they are adopted as well for PRISM, ESMF, and other international projects. The tables can be found at: http://www.cgd.ucar.edu/cms/eaton/cf-metadata/index.html

1. Base names:
   A variable X has a base name `x`, which can be modified with a prefix and a suffix.
2. Prefix:
   - for derivatives:
     - time derivative $\partial X/\partial t$: `ddt_x`
     - derivative in normal direction of edge of triangular cell $\partial X/\partial xn$: `ddxn_x`
     - derivative in tangential direction $\partial X/\partial xt$: `ddxt_x`
     - vertical derivative $\partial X/\partial z$: `ddz_x`
   - for fluxes:
     - `fl_`
   - for DOCTOR standard:
     - e.g. `REAL` argument `x` in parameter list: `p_x`
3. Suffix: (use economically for clarity)
   - for tendencies related to a specific process:
     - e.g. $\partial T/\partial t|_{\text{radiation}}$: `ddt_T_rad`
   - as designator of position at which the variable is defined
     - in vertex of Delaunay grid: `_vd`
     - in vertex of Voronoi cell: `_vv`
     - on edge of Delaunay grid: `_ed`
     - on edge of Voronoi cell: `_ev`
     - lower boundary: `_lb`
     - upper boundary: `_ub`
     - ...
   - other labels to be specified
     - 2m: `_2m`
     - surface: `_sfc`
     - snow: `_snow`
     - ice: `_ice`
     - ...

Section G of the appendix provides a list of global variable names.
Lists are not finalized yet. They should be updated as soon as decisions are made.

6. Program structure

In general routines should not exceed 50-100 lines (`FUNCTION`s or `SUBROUTINE`s) and each programming unit should begin with a header explaining the given sections. The header and variable declaration sections allows as well for automatic processing to include information in a documentation either of libraries or in the models description. Templates are provided in the appendix.

7. Literature

Kalnay, E. et al., *Rules for Interchange of Physical Parametrizations*, Bull. A.M.S., 70 No. 6, p 620, **1989**.

Panskus, Heiko, and Heinke Schlünzen, *Standards und Richtlinien für MITRAS/METRAS*, Institut für Meteorologie, Universität Hamburg, **1997**.

Andrews, Phillip (UKMO), Gerard Cats (KNMI/HIRLAM), David Dent (ECMWF), Michael Gertz (DWD), and Jean Louis Ricard (Meteo France), *European Standards For Writing and Documenting Exchangeable Fortran 90 Code, Version 1.1*, **1995**.
`http://www.meto.gov.uk/research/nwp/numerical/fortran90/f90_standards.html`

Regionales Rechenzentrum für Niedersachsen/Universität Hannover, *Fortran 95 für ANSI-Fortran / DIN-Fortran / ISO/IEC 1539-1*,**1997**.

DKRZ-Model User Support Group, *ECHAM3 - Atmospheric General Circulation Model*, Deutsches Klimarechenzentrum, Report Nr.6, Hamburg, **1992**.
`http://www.MPI-Met.mpg.de/en/extra/models/echam/echam3_DKRZ-ReportNo.6.pdf`

Sawyer, William and Arlindo da Silva *ProTEX: A Sample Fortran 90 Source Code Documentation System*, Goddard Space Flight Center, Data Assimilation Office, DAO Office Note 97-11, Greenbelt, Maryland, **1997**.
`http://gmao.gsfc.nasa.gov/software/protex/protex.php`

Part II.
Appendix

A. Prologue for main program

```
PROGRAM <name>
!-------------------------------------------------------------------------
!
!    ProTeX FORTRAN source: Style 2
!    modified for ICON project, DWD/MPI-M 2005
!
!-------------------------------------------------------------------------
!BOI
!
! !TITLE: <title>
!
! !AUTHORS: <author1, author2>
!
! !AFFILIATION: <affiliation1>
!
! !AUTHORS: <author3>
!
! !AFFILIATION: <affiliation2>
!
! !DATE: <\today or fixed date>
!
!EOI
!-------------------------------------------------------------------------
!BOP
!
! !ROUTINE: <name> --- main program
!
! !DESCRIPTION:
! <Describe the function of the routine and algorithm(s) used in
! the routine. Include any applicable external references.>
!
! !REVISION HISTORY:
! <Description of activity> by <Name> (YYYY-<MM-DD>)
!
! !COPYRIGHT:
! 2002-2005 by DWD and MPI-M
! This software is provided for non-commercial use only.
! See the LICENSE and the WARRANTY conditions.
!
! !LICENSE:
! The use of ICON is hereby granted free of charge for an unlimited time,
! provided the following rules are accepted and applied:
!
! 1. You may use or modify this code for your own non commercial and non
!    violent purposes.
! 2. The code may not be re-distributed without the consent of the authors.
! 3. The copyright notice and statement of authorship must appear in all
```

```
!    copies.
! 4. You accept the warranty conditions (see WARRANTY).
! 5. In case you intend to use the code commercially, we oblige you to sign
!    an according license agreement with DWD and MPI-M.
!
! !WARRANTY:
! This code has been tested up to a certain level. Defects and weaknesses,
! which may be included in the code, do not establish any warranties by the
! authors.
! The authors do not make any warranty, express or implied, or assume any
! liability or responsibility for the use, acquisition or application of this
! software.
! --- End of Copyright/License/Warranty
!
! !USES:
  USE <module>

  IMPLICIT NONE

! !VERSION CONTROL:
  CHARACTER(len=*), PARAMETER :: version = '$Id$'

!
!EOP
!-----------------------------------------------------------------------
!BOC
```

B. Prologue for modules

B.1. Modules containing data and parameters

```
MODULE mo_xyz
!-------------------------------------------------------------------------
!
!    ProTeX FORTRAN source: Style 2
!    modified for ICON project, DWD/MPI-M 2005
!
!-------------------------------------------------------------------------
!BOP
!
! !MODULE: <Module name>
!
! !DESCRIPTION:
! <Describe the function of the routine and algorithm(s) used in
! the routine. Include any applicable external references.>
!
! !REVISION HISTORY:
! <Description of activity> by <Name> (YYYY-<MM-DD>)
!
! !COPYRIGHT:
! 2002-2005 by DWD and MPI-M
! This software is provided for non-commercial use only.
! See the LICENSE and the WARRANTY conditions.
!
```

```
! !LICENSE:
! The use of ICON is hereby granted free of charge for an unlimited time,
! provided the following rules are accepted and applied:
!
! 1. You may use or modify this code for your own non commercial and non
!    violent purposes.
! 2. The code may not be re-distributed without the consent of the authors.
! 3. The copyright notice and statement of authorship must appear in all
!    copies.
! 4. You accept the warranty conditions (see WARRANTY).
! 5. In case you intend to use the code commercially, we oblige you to sign
!    an according license agreement with DWD and MPI-M.
!
! !WARRANTY:
! This code has been tested up to a certain level. Defects and weaknesses,
! which may be included in the code, do not establish any warranties by the
! authors.
! The authors do not make any warranty, express or implied, or assume any
! liability or responsibility for the use, acquisition or application of this
! software.
! --- End of Copyright/License/Warranty
!
! !USES:
  USE <module>

  IMPLICIT NONE

! !VERSION CONTROL:
  CHARACTER(len=*), PARAMETER :: version = '$Id$'

! !DEFINED PARAMETERS:
  <type>, PARAMETER     :: <parameter>     ! <Parameter description>

! !GLOBAL VARIABLES:
  <type>                :: <variable>      ! <Variable description>

!EOP
!-----------------------------------------------------------------------
!BOC

END MODULE mo_xyz

!EOC
```

B.2. Modules containing subroutines and functions

```
MODULE m_xyz
!-----------------------------------------------------------------------
!
!    ProTeX FORTRAN source: Style 2
!    modified for ICON project, DWD/MPI-M 2005
!
!-----------------------------------------------------------------------
!BOP
!
```

```
! !MODULE: <Module name>
!
! !DESCRIPTION:
! <Describe the function of the routine and algorithm(s) used in
! the routine. Include any applicable external references.>
!
! !REVISION HISTORY:
! <Description of activity> by <Name> (YYYY-<MM-DD>)
!
! !COPYRIGHT:
! 2002-2005 by DWD and MPI-M
! This software is provided for non-commercial use only.
! See the LICENSE and the WARRANTY conditions.
!
! !LICENSE:
! The use of ICON is hereby granted free of charge for an unlimited time,
! provided the following rules are accepted and applied:
!
! 1. You may use or modify this code for your own non commercial and non
!    violent purposes.
! 2. The code may not be re-distributed without the consent of the authors.
! 3. The copyright notice and statement of authorship must appear in all
!    copies.
! 4. You accept the warranty conditions (see WARRANTY).
! 5. In case you intend to use the code commercially, we oblige you to sign
!    an according license agreement with DWD and MPI-M.
!
! !WARRANTY:
! This code has been tested up to a certain level. Defects and weaknesses,
! which may be included in the code, do not establish any warranties by the
! authors.
! The authors do not make any warranty, express or implied, or assume any
! liability or responsibility for the use, acquisition or application of this
! software.
! --- End of Copyright/License/Warranty
!
! !USES:
  USE <module>

  IMPLICIT NONE

! !VERSION CONTROL:
  CHARACTER(len=*), PARAMETER :: version = '$Id$'

! !PRIVATE TYPES:
  <type declaration>                        ! <Type description>

! !PRIVATE MEMBER FUNCTIONS:
  <function>                                ! <Function description>

! !PUBLIC TYPES:
  <type declaration>                        ! <Type description>

! !PUBLIC MEMBER FUNCTIONS:
  <function>                                ! <Function description>
```

```
! !PUBLIC DATA MEMBERS:
  <type>                :: <variable>      ! <Variable description>

! !DEFINED PARAMETERS:
  <type>, PARAMETER     :: <parameter>     ! <Parameter description>

! !LOCAL VARIABLES:
  <type>                :: <variable>      ! <Variable description>

CONTAINS

!EOP
!------------------------------------------------------------------------
!BOC
!
!EOC
```

C. Prologue for functions

```
!------------------------------------------------------------------------
!BOP
!
! !IROUTINE: <Function name>
!
! !FUNCTION INTERFACE:
FUNCTION <name> (<arguments>) ( RESULT (name_result) )
!
! !DESCRIPTION:
! <Describe the function of the routine and algorithm(s) used in
! the routine. Include any applicable external references.>
!
! !REVISION HISTORY:
! <Description of activity> by <Name> (YYYY-<MM-DD>)
!
! !USES:
  USE <module>

! !DEFINED PARAMETERS:
  <type>, PARAMETER     :: <parameter>     ! <Parameter description>

! !INPUT PARAMETERS:
  <type>, INTENT(in)    :: <parameter>     ! <Parameter description>

! !INPUT/OUTPUT PARAMETERS:
  <type>, INTENT(inout) :: <parameter>     ! <Parameter description>

! !RETURN VALUE:
  <type>                :: <name_result>   ! <Return value description>

! !OUTPUT PARAMETERS:
  <type>, INTENT(out)   :: <parameter>     ! <Parameter description>

! !LOCAL VARIABLES:
  <type>                :: <variable>      ! <Variable description>
```

```
!EOP
!-------------------------------------------------------------------------
!BOC
```

D. Prologue for subroutines

```
!-------------------------------------------------------------------------
!BOP
!
! !IROUTINE: <Subroutine name>
!
! !SUBROUTINE INTERFACE:
SUBROUTINE <name> (<arguments>)
!
! !DESCRIPTION:
! <Describe the function of the routine and algorithm(s) used in
! the routine. Include any applicable external references.>
!
! !REVISION HISTORY:
! <Description of activity> by <Name> (YYYY-<MM-DD>)
!
! !USES:
  USE <module>

! !DEFINED PARAMETERS:
  <type>, PARAMETER     :: <parameter>     ! <Parameter description>

! !INPUT PARAMETERS:
  <type>, INTENT(in)    :: <parameter>     ! <Parameter description>

! !INPUT/OUTPUT PARAMETERS:
  <type>, INTENT(inout) :: <parameter>     ! <Parameter description>

! !OUTPUT PARAMETERS:
  <type>, INTENT(out)   :: <parameter>     ! <Parameter description>

! !LOCAL VARIABLES:
  <type>                :: <variable>      ! <Variable description>

!EOP
!-------------------------------------------------------------------------
!BOC
```

E. mo_kind

```
MODULE mo_kind
!-------------------------------------------------------------------------
!
!    ProTeX FORTRAN source: Style 2
!    modified for ICON project, DWD/MPI-M 2005
!
!-------------------------------------------------------------------------
```

```
!BOP
!
! !MODULE:  mo_kind
!
! !DESCRIPTION:
!  Module determines kinds for different precisions
!
!  Number model from which the SELECTED\_*\_KIND are requested: \\
!
!\begin{tabular}{r@{\hspace*{3em}}c@{\hspace*{3em}}c}
!                      &4 byte REAL     &8 byte REAL        \\
!        CRAY:         &-               &precision =   13   \\
!                      &                &exponent  = 2465   \\
!        IEEE:         &precision = 6   &precision =   15   \\
!                      &exponent  = 37  &exponent  =  307
!\end{tabular}
!\medskip
!
!  Most likely this are the only possible models.
!
! !REVISION HISTORY:
!  Working precision and comments are added by Luis Kornblueh (2001)
!  Working precision are moved to mo_constants by Luca Bonaventura (2003)
!  Modified to ProTeX-style by Thomas Heinze (2004).
!
! !COPYRIGHT:
! 2002-2005 by DWD and MPI-M
! This software is provided for non-commercial use only.
! See the LICENSE and the WARRANTY conditions.
!
! !LICENSE:
! The use of ICON is hereby granted free of charge for an unlimited time,
! provided the following rules are accepted and applied:
!
! 1. You may use or modify this code for your own non commercial and non
!    violent purposes.
! 2. The code may not be re-distributed without the consent of the authors.
! 3. The copyright notice and statement of authorship must appear in all
!    copies.
! 4. You accept the warranty conditions (see WARRANTY).
! 5. In case you intend to use the code commercially, we oblige you to sign
!    an according license agreement with DWD and MPI-M.
!
! !WARRANTY:
! This code has been tested up to a certain level. Defects and weaknesses,
! which may be included in the code, do not establish any warranties by the
! authors.
! The authors do not make any warranty, express or implied, or assume any
! liability or responsibility for the use, acquisition or application of this
! software.
! --- End of Copyright/License/Warranty

  IMPLICIT NONE

! !VERSION CONTROL:
  CHARACTER(len=*), PARAMETER :: version = '$Id$'
```

```
! !DEFINED PARAMETERS:

!
! ! Floating point section
!

  INTEGER, PARAMETER :: sp = SELECTED_REAL_KIND(6,37)
  INTEGER, PARAMETER :: dp = SELECTED_REAL_KIND(12,307)
  INTEGER, PARAMETER :: qp = SELECTED_REAL_KIND(12,307)

  INTEGER, PARAMETER :: wp = SELECTED_REAL_KIND(12,307)

!
! ! Integer section
!

  INTEGER, PARAMETER :: i4 = SELECTED_INT_KIND(9)
  INTEGER, PARAMETER :: i8 = SELECTED_INT_KIND(14)

!EOP
!------------------------------------------------------------------------
!BOC

END MODULE mo_kind

!EOC
```

F. Constants

F.1. Mathematical constants

```
MODULE mo_math_constants
!------------------------------------------------------------------------------
!
!    ProTeX FORTRAN source: Style 2
!    modified for ICON project, DWD/MPI-M 2005
!
!------------------------------------------------------------------------------
!BOP
!
! !MODULE:  mo_math_constants\\
!
! !DESCRIPTION:
!  Module determines main mathematical constants to be used by shallow water
!  prototype. Values are taken from glibc 2.2.5: /usr/include/math.h
!  These constants
!  are provided to more significant digits than is necessary for a 64-bit
!  double precision number; they may be used for other purposes where the
!  extra precision is necessary or useful.
!
! !REVISION HISTORY:
!  Developed  by Luis Kornblueh (2004)
!  Modified to ProTeX-style by  Luca Bonaventura and Thomas Heinze (2004).
```

```
!  Modified according to style guide by Thomas Heinze (2005-06-24):
!   - module renamed from mo_math to mo_math_constants
!   - eps moved from mo_physical_constants
!   - pid180i renamed to convfact
!  Including some more constants from math.h by Thomas Heinze (2005-07-18)
!
!
! !COPYRIGHT:
! 2002-2005 by DWD and MPI-M
! This software is provided for non-commercial use only.
! See the LICENSE and the WARRANTY conditions.
!
! !LICENSE:
! The use of ICON is hereby granted free of charge for an unlimited time,
! provided the following rules are accepted and applied:
!
! 1. You may use or modify this code for your own non commercial and non
!    violent purposes.
! 2. The code may not be re-distributed without the consent of the authors.
! 3. The copyright notice and statement of authorship must appear in all
!    copies.
! 4. You accept the warranty conditions (see WARRANTY).
! 5. In case you intend to Modified according to style guideuse the code commercially
!    an according license agreement with DWD and MPI-M.
!
! !WARRANTY:
! This code has been tested up to a certain level. Defects and weaknesses,
! which may be included in the code, do not establish any warranties by the
! authors.
! The authors do not make any warranty, express or implied, or assume any
! liability or responsibility for the use, acquisition or application of this
! software.
! --- End of Copyright/License/Warranty
!
! !USES:
USE mo_kind,      ONLY:  wp

IMPLICIT NONE

! !VERSION CONTROL:
  CHARACTER(len=*), PARAMETER :: version = '$Id$'

! !DEFINED PARAMETERS:

!
! ! Mathematical constants (in brackets original C names)
!
! ! euler     (M_E       )  -- e
! ! log2e     (M_LOG2E   )  -- log2(e)
! ! log10e    (M_LOG10E  )  -- log10(e)
! ! ln2       (M_LN2     )  -- ln(2)
! ! ln10      (M_LN10    )  -- ln(10)
! ! pi        (M_PI      )  -- pi
! ! pi_2      (M_PI_2    )  -- pi/2
! ! pi_4      (M_PI_4    )  -- pi/4
! ! rpi       (M_1_PI    )  -- 1/pi
```

```
! ! rpi_2     (M_2_PI    )  -- 2/pi
! ! rsqrtpi_2 (M_2_SQRTPI)  -- 2/(sqrt(pi))
! ! sqrt2     (M_SQRT2   )  -- sqrt(2)
! ! sqrt1_2   (M_SQRT1_2 )  -- 1/sqrt(2)
! !
!
!

REAL (wp), PARAMETER ::  euler     = 2.71828182845904523536028747135266250_wp
REAL (wp), PARAMETER ::  log2e     = 1.44269504088896340735992468100189214_wp
REAL (wp), PARAMETER ::  log10e    = 0.43429448190325182765112891891660508 2_wp
REAL (wp), PARAMETER ::  ln2       = 0.69314718055994530941723212145817656 8_wp
REAL (wp), PARAMETER ::  ln10      = 2.30258509299404568401799145468436421_wp
REAL (wp), PARAMETER ::  pi        = 3.14159265358979323846264338327950288_wp
REAL (wp), PARAMETER ::  pi_2      = 1.57079632679489661923132169163975144_wp
REAL (wp), PARAMETER ::  pi_4      = 0.78539816339744830961566084581987572 1_wp
REAL (wp), PARAMETER ::  rpi       = 0.31830988618379067153776752674502872 4_wp
REAL (wp), PARAMETER ::  rpi_2     = 0.63661977236758134307553505349005744 8_wp
REAL (wp), PARAMETER ::  rsqrtpi_2 = 1.12837916709551257389615890312154517_wp
REAL (wp), PARAMETER ::  sqrt2     = 1.41421356237309504880168872420969808_wp
REAL (wp), PARAMETER ::  sqrt1_2   = 0.70710678118654752440084436210484903 9_wp

!
! ! some more useful constants
! ! pi_5    -- half angle of pentagon
! ! rad2deg -- conversion factor from radians to degree
! ! eps     -- residual bound for solvers
!

REAL (wp), PARAMETER ::  pi_5      = pi*0.2_wp
REAL (wp), PARAMETER ::  rad2deg   = 180.0_wp/pi
REAL (wp), PARAMETER ::  eps       = 1.e-8_wp

!
! ! phi0 is  the latitude of the lowest major triangle corner
! ! and the latitude of the major hexagonal faces centers
! ! phi0 = 0.5_wp*pi - 2._wp*acos(1.0_wp/(2._wp*sin(pi/5._wp)))
!

REAL (wp), PARAMETER :: phi0  = 0.46364760900080614903_wp


!EOP
!-----------------------------------------------------------------------
!BOC

END MODULE mo_math_constants

!EOC
```

F.2. Physical constants

```
MODULE mo_physical_constants
!-----------------------------------------------------------------------
!
```

```
!    ProTeX FORTRAN source: Style 2
!    modified for ICON project, DWD/MPI-M 2005
!
!-------------------------------------------------------------------------
!BOP
!
! !MODULE: mo_physical_constants\\
!
! !DESCRIPTION:
!  Module determines physical constants to be used by shallow water prototype.
!  Values are taken from Williamson et al (1992).
!
! !REVISION HISTORY:
!  Developed  by Luis Kornblueh and Luca Bonaventura (2002-3)
!  Modified to ProTeX-style by  Luca Bonaventura and Thomas Heinze (2004).
!  Modified according to style guide by Thomas Heinze (2005-06-24):
!   - module renamed from mo_constants to mo_physical_constants
!   - eps moved to mo_math_constants
!   - su0 renamed to u0 (as in Williamson et al. (1992) paper)
!
! !COPYRIGHT:
! 2002-2005 by DWD and MPI-M
! This software is provided for non-commercial use only.
! See the LICENSE and the WARRANTY conditions.
!
! !LICENSE:
! The use of ICON is hereby granted free of charge for an unlimited time,
! provided the following rules are accepted and applied:
!
! 1. You may use or modify this code for your own non commercial and non
!    violent purposes.
! 2. The code may not be re-distributed without the consent of the authors.
! 3. The copyright notice and statement of authorship must appear in all
!    copies.
! 4. You accept the warranty conditions (see WARRANTY).
! 5. In case you intend to use the code commercially, we oblige you to sign
!    an according license agreement with DWD and MPI-M.
!
! !WARRANTY:
! This code has been tested up to a certain level. Defects and weaknesses,
! which may be included in the code, do not establish any warranties by the
! authors.
! The authors do not make any warranty, express or implied, or assume any
! liability or responsibility for the use, acquisition or application of this
! software.
! --- End of Copyright/License/Warranty
!
! !USES:

USE mo_kind,            ONLY: wp
USE mo_math_constants,  ONLY: pi

IMPLICIT NONE

! !VERSION CONTROL:
  CHARACTER(len=*), PARAMETER :: version = '$Id$'
```

```
! !DEFINED PARAMETERS:

! ! Physical constants
!

REAL (wp), PARAMETER :: re      = 6.37122e6_wp ! av. radius of the earth [m]
REAL (wp), PARAMETER :: rre     = 1._wp/re
REAL (wp), PARAMETER :: grav    = 9.80616_wp   ! av. gravitational const.[m/s^2]
REAL (wp), PARAMETER :: rgrav   = 1._wp/grav
REAL (wp), PARAMETER :: omega   = 7.292e-5_wp  ! angular velocity of earth [1/s]

!
! ! Parameters used often by test cases
!

REAL (wp), PARAMETER :: u0 =(2.0_wp*pi*re)/(12.0_wp*24.0_wp*3600.0_wp) ! [m/s]

!EOP
!-----------------------------------------------------------------------
!BOC

END MODULE mo_physical_constants

!EOC
```

G. Global variable names

see remarks about these tables in section 5 on page 11.

G.1. Mathematical and physical constants

see section F of appendix on page 20

G.2. Fields that are constant in time

name	description	phys.dim.
f	Coriolis parameter	$1/s$
geo_sfc	orography * grav	m^2/s^2

G.3. Prognostic single level fields

name	description	phys.dim.
temp_sfc	surface temperature	K
temp_snow	temperature at the top of snow or surface temperature (if no snow)	K

G.4. Atmospheric fields

name	description	phys.dim.
`cpair`	heat capacity at constant pressure	J/Kkg
`cvair`	heat capacity at constant volume	J/Kkg
`div`	divergence	$1/s$
`geo`	geopotential	m^2/s^2
`mass`	total mass	kg
`massX`	partial mass of component X, e.g. H_2O `massh2o`	kg/m^3
`pres`	pressure	Pa
`rho`	total density	kg/m^3
`rhoX`	partial density of component X, e.g. H_2O `rhoh2o`	kg/m^3
`temp`	temperature	K
`tempv`	virtual temperature	K
`u`	reconstructed velocity in zonal direction	m/s
`v`	reconstructed velocity in meridional direction	m/s
`vn`	normal velocity across triangle edge	m/s
`vort`	vorticity	$1/s$
`vt`	tangential velocity along triangle edge	m/s
`w`	vertical velocity in height system	m/s
`wind`	3d velocity vector	m/s
`wpres`	vertical velocity in pressure system (ω)	Pa/s

G.5. Horizontal grid

name	description	phys.dim.
`area`	area	m^2
`lat`	geographical latitude of a gridpoint	rad
`lon`	geographical longitude of a gridpoint	rad