Coding standards for Arpège, IFS and Aladin Systems

Météo-France/GMAP and ECMWF
contacts: claude.fischer@meteo.fr, mike.fisher@ecmwf.int

25 novembre 2011

This document summarizes the most important coding norms and rules one has to keep in mind while designing and writing a piece of code in the Arpège-IFS software.
This note supersedes the previous document about coding norms, which is kept available for traceback only at : http ://www.cnrm.meteo.fr/gmapdoc/IMG/pdf/tutorial.pdf

1 Standards for the presentation of the code

1. **Header**
   Procedures must start with a documentation header in English containing, in this exact order :
   - A brief description of the procedure.
   - Interface details describing all dummy arguments, in same order as they appear in the call to the procedure.
   - A list of all externals called by the procedure.
   - A description of the method or a reference to external documentation.
   - The name of the author and the creation date of the procedure.
   - A list of all modifications to the procedure made during the last 5 years.
2. **Variable Declarations**
   Variables must be declared just after declaration header, in the following order :
   - Module variable declarations.
   - Dummy arguments declarations (with INTENT) in the same order as in the call to the procedure.
   - Local variables declarations.
   - Function declarations.
   - Interface blocks.

   Variables should preferably be declared separately (in order to facilitate merging) and grouped according to type and attributes with separating commas at the end of the line.

   In USE lists, the items have to be presented in an easily readable manner.
3. **Code Body**
   The code body must be split in sections and subsections, which should be clearly separated from each other.

   Lines should be broken in a readable manner with no more than 132 characters per line.

   For readability, nesting of conditional blocks should not be more than 3 levels deep.

   Where blocks are deeply nested, long or complex, they should be given character labels.
4. **Indentation Rules**
   Code starts at column 1 except within conditional blocks and DO loops. Within such blocks, code must be indented with 2 blank spaces with a further 2 blank spaces for each level of nesting.
5. **Naming Schemes for Modules**
   All new modules should end with _mod. The name of the file should match

the name of the module it contains (e.g. MODULE EINT should be in file eint_mod.F90)

6. In calls, optional arguments must be labelled, not resolved by position.
7. **Continuation Lines**
   An ampersand (&) followed by an identation is mandatory for continuation lines. The indentation should assist the readability of the statement.
8. Long CALL sequences should be broken at the same places as the continuation marks in the SUBROUTINE sequence in order to ease the reading of long list of dummy arguments.
9. The first and last executable statements in a subroutine must be calls to DR_HOOK. The string argument to these calls must give the name of subroutine. In the case of a contained subroutine, the string should be constructed from the name of the parent routine and the contained routine, separated with a percent symbol (%).
10. END statement must include the name of the subroutine.

Figure 1 shows a header documentation for a procedure.

FIG. 1 – Example of header documentation and variables declarations in a procedure.

```
SUBROUTINE CODESTY(LDLAM,KERR)

! Purpose :
! -------
!   *CODESTY* : CODE STYle : To show coding standards in Arpege/Ifs/Aladin.

! Interface :
! ---------

! Input :
! -----
!   LDLAM  : key to identify model type

! Output :
! ------
!   KERR   : error code of the subroutine

! Externals :
! ---------
!   None.

! Method :
! ------

! Reference :
! ---------
!   Coding standards in Arpege/Ifs/Aladin.

! Author :
! ------
!   19-Jul-2002 Ryad El Khatib        *METEO-FRANCE*
```

```
! Modifications :
! -------------
!   30-Oct-2003 M. Hamrud              Cleaning for Cycle 28
!   30-Feb-0000 R. Randriamampianina  Remove comments that were in Hungarian
!   21-Oct-2011 C. Fischer            Add some body statements
! End Modifications
!-----------------------------------------------------------------------

USE PARKIND1 , ONLY : JPIM,    JPRB
USE YOMHOOK   ,ONLY : LHOOK,   DR_HOOK
USE YOMCT0   , ONLY : LELAM
USE YOMLUN   , ONLY : NULOUT

!-----------------------------------------------------------------------

IMPLICIT NONE

LOGICAL, INTENT(IN) :: LDLAM
INTEGER(KIND=JPIM), INTENT(OUT) :: KERR

!-----------------------------------------------------------------------

INTEGER(KIND=JPIM), PARAMETER :: JPLEN=16  ! Length of the local message
CHARACTER(LEN=JPLEN) :: CLMESS    ! A local message
REAL(KIND=JPRB) :: ZHOOK_HANDLE

!-----------------------------------------------------------------------

#include "ejust_do_it.intfb.h"
#include "just_do_it.intfb.h"

!-----------------------------------------------------------------------
```

```
IF (LHOOK) CALL DR_HOOK('CODESTY',0,ZHOOK_HANDLE)

!-----------------------------------------------------------------------

!* 1.  CHECK WHAT MODEL THIS IS

IF (LDLAM) THEN
  WRITE(NULOUT,*) ' THIS IS A LIMITED AREA RUN ', &
   & 'AND THESE GUYS ARE NICE'
  CLMESS='ALADIN MODEL'
  CALL EJUST_DO_IT(CLMESS)
ELSE
  WRITE(NULOUT,*) ' THIS IS A GLOBAL RUN ', &
   & 'AND THESE GUYS ARE UGGLY'
  CLMESS='IFS/ARPEGE MODEL'
  CALL JUST_DO_IT(CLMESS)
ENDIF

!-----------------------------------------------------------------------

!* 2.  SET ERROR PARAMETER FOR RETURN

KERR=0_JPIM

IF ( (LDLAM.AND.(.NOT.LELAM)) .OR. ((.NOT.LDLAM).AND.LELAM) ) KERR=1_JPIM

!-----------------------------------------------------------------------

IF (LHOOK) CALL DR_HOOK('CODESTY',1,ZHOOK_HANDLE)

END SUBROUTINE CODESTY
```

2 Preliminary design of the code

1. Encapsulation rules.
   Modules should be splitted up in a sensible manner to avoid too long Fortran files or too complex modules. One recommandation can be to separate the data structures, the operators and the descriptive parameters (including setup if existing). The number of entities in a single module is not limited, but a reasonable total number should be considered always (about 10 to 20 entities ?).
2. Subroutines should have no more than 300 executable statements. For a module containing several entities, this limit of 300 executable statements is applicable for each inner subroutine.
3. Avoid cosmetic changes that will make merges difficult (such as re-ordering argument lists and USE statements, or changing the indentation of large blocks of code). As exception, cosmetic changes can happen when a routine is heavily modified, and only one well identified developer will contribute to the code for the next common cycle.
4. Declarations of unused variables must be removed
5. Variables suffixed with `L` are local in the sense of the parallel distribution. Variables suffixed with `G` are global.
6. The use of array syntax is not recommended except for initalization and very basic computations.
7. Cut-and-paste of existing piece of code should be avoided. Common code should be extracted to a separate subroutine or function.
8. The variable `LECMWF` should be used only in setup subroutines.
9. The variable `LELAM` is not to be used below SCAN2M.
10. The choice between LFI/LFA or GRIB format should be made using the variables LARPEGEF or LARPEGEF_xx (and not LECMWF).
11. The MPL package must be used as the interface for any message passing.
12. Derived types should be declared in a module.
13. Code must be threadsafe.

3 Detailed design of the code

1. Abnormal termination must be invoked by ABOR1.
2. Variables in data modules must be saved using the SAVE statement.
3. Array shape and Variable type must not be changed when passed to a subroutine.
4. Use SELECT CASE when possible instead of IF/ELSEIF/ELSE/ENDIF.
5. For each called routine there must be a "`"#include"` statement that includes an explicit interface block for the routine. Note that the files containing the explicit interface blocks are automatically generated during compilation.
6. Routines should have a small number of dummy arguments. Routine with more than 50 dummy arguments are not allowed.
7. Variable names should be meaningful to an English reader. Very short names should be reserved for loop indices.
8. Conventional prefixes or suffixes are to be used for all variables except derived types, as described in table 1 in section 4. There is no naming convention for derived types.
9. Aladin subroutines that are counterparts of IFS/Arpège ones should have the same name but prefixed with E. Aladin setup routines that are counterparts of IFS/Arpège (prefixed SU) should be prefixed SUE.
10. The logical unit for output listing is NULOUT. Output to NULOUT must be deterministic and should not change according to the parallel distribution or the time at which the job is run. Error messages should be written to unit NULERR.
11. Universal constants must be stored, saved and initialized in data module YOMCST. They cannot be modified elsewhere and should not be accessed via dummy arguments.
12. Calls to MPL subroutines should provide a CDSTRING identifying the caller.
13. Source code is partitioned into projects. Each source file must be put in the proper directory for its project.
14. Runtime specification of variables must be done using namelists.
15. DATA statement should be avoided if possible and is allowed only for small lists.

4 Detailed Fortran coding standards

1. The code should be Fortran 90 free format.
2. Use a consistent style throughout each module and subroutine.
3. The TAB character is not allowed.
4. `IMPLICIT NONE` is mandatory in all routines.
5. Array dimensions must not be hard-coded.
6. Declarations must use the notation "`::`".
7. Variables and constants must be declared with explicit kind, using the kinds defined in PARKIND1 and PARKIND2.
8. All USE statements must include an "ONLY" clause, except for modules that override ASSIGNMENT, where this is dangerous.
9. Constants should be PARAMETERs wherever as possible
10. Variable names should follow the prefix convention defined in table 1.

Status or scope / Type	Variable in data module	Dummy argument	Local variable	Loop control	Any Parameter
INTEGER	M, N	K	I	J but not JP	JP
REAL	A, B, E to H, O, Q to X	P but not PP	Z	-	PP
LOGICAL	L but not (LD,LL,LP)	LD	LL	-	LP
CHARACTER	C but not (CD,CL,CP)	CD	CL	-	CP

Tab. 1 – Naming Convention for Variables

11. The following statements are banned :
    (a) `STOP`
    (b) `PRINT`
    (c) `RETURN`
    (d) `ENTRY`
    (e) `DIMENSION`
    (f) `DOUBLE PRECISION`
    (g) `COMPLEX`
    (h) `GO TO`
    (i) `CONTINUE`
    (j) `FORMAT`

(k) `COMMON`

(l) `EQUIVALENCE`

12. Arrays should not be declared with implicit size : "A(*)".
13. Large arrays should be allocatable. Small or low-level arrays should be automatic.
14. All allocated arrays should be explicitly deallocated.
15. Use Fortran 90 comparison operators (e.g. == rather than .EQ.).
16. Explicitly set variables (parameters, constants, namelist variables,...) should be always exactly compared (using `==` or `\=`, etc). Evaluated variables (that might be subject to roundoff error) should be tested against a reference using a threshold.
17. All dummy arguments must specify the `INTENT` attribute
18. Optional arguments must be called in the same order they are declared.
19. END statements for blocks should not have a space after END. For example an IF block should end with ENDIF, not "END IF".
20. Inactive (e.g. commented-out) code must be removed. (However, it is acceptable for explanatory comments to include example code.)