CODING NORMS IN ARPEGE/ALADIN (CY36T2).

K. YESSAD
METEO-FRANCE/CNRM/GMAP/ALGO

DOCUMENTATION.

- El Khatib, R., 2003 : Coding standards for ARPEGE/IFS/ALADIN.
- Yessad, K., 2009 : Library architecture and history of the technical aspects in ARPEGE/IFS, ALADIN and AROME in the cycle 36 of ARPEGE/IFS.
- ? ? ?, 2009 : Fortran coding standards for the ARPEGE, IFS and ALADIN systems (provisional draft, 8pp).

DOCUMENTATION.

- El Khatib, R., 2003 : Coding standards for ARPEGE/IFS/ALADIN.
- Yessad, K., 2009 : Library architecture and history of the technical aspects in ARPEGE/IFS, ALADIN and AROME in the cycle 36 of ARPEGE/IFS.
- ? ? ?, 2009 : Fortran coding standards for the ARPEGE, IFS and ALADIN systems (provisional draft, 8pp).

WHY CODING NORMS ?

- Around 13000 routines (3500 routines in ARP).
- Around 4 million code lines (1.6 million in ARP).
- Around 200000 additional code lines per year.
- Around 100 people (probably more) of several countries working on the same code.
- Dirty code :
  - can be not understandable.
  - more bugs ; more time to debug.
  - more difficult to develop on it.
- Some of these rules (but not all) date from the beginning of the ARPEGE project.
- Revision expected in the OOPS project but most rules will remain valid.

WHY CODING NORMS ?

- Around 13000 routines (3500 routines in ARP).
- Around 4 million code lines (1.6 million in ARP).
- Around 200000 additional code lines per year.
- Around 100 people (probably more) of several countries working on the same code.
- Dirty code :
  - can be not understandable.
  - more bugs ; more time to debug.
  - more difficult to develop on it.
- Some of these rules (but not all) date from the beginning of the ARPEGE project.
- Revision expected in the OOPS project but most rules will remain valid.

WHY CODING NORMS ?

- Around 13000 routines (3500 routines in ARP).
- Around 4 million code lines (1.6 million in ARP).
- Around 200000 additional code lines per year.
- Around 100 people (probably more) of several countries working on the same code.
- Dirty code :
  - can be not understandable.
  - more bugs ; more time to debug.
  - more difficult to develop on it.
- Some of these rules (but not all) date from the beginning of the ARPEGE project.
- Revision expected in the OOPS project but most rules will remain valid.

WHY CODING NORMS ?

- Around 13000 routines (3500 routines in ARP).
- Around 4 million code lines (1.6 million in ARP).
- Around 200000 additional code lines per year.
- Around 100 people (probably more) of several countries working on the same code.
- Dirty code :
  - can be not understandable.
  - more bugs ; more time to debug.
  - more difficult to develop on it.
- Some of these rules (but not all) date from the beginning of the ARPEGE project.
- Revision expected in the OOPS project but most rules will remain valid.

WHY CODING NORMS ?

- Around 13000 routines (3500 routines in ARP).
- Around 4 million code lines (1.6 million in ARP).
- Around 200000 additional code lines per year.
- Around 100 people (probably more) of several countries working on the same code.
- Dirty code :
  - can be not understandable.
  - more bugs ; more time to debug.
  - more difficult to develop on it.
- Some of these rules (but not all) date from the beginning of the ARPEGE project.
- Revision expected in the OOPS project but most rules will remain valid.

A WELL CODED ROUTINE.

```
SUBROUTINE FPTRATOD(KC,KFLDS,PFIELD_ARR,PFIELD_DEP)


!**** *FPTRATOD*  - FULL-POS TRANSPOSITION.
!                   Arrival geometry DM-distribution towards
!                   departure geometry DM-distribution.

!     PURPOSE.
!     --------

!**   INTERFACE.
!     ----------
!       *CALL* *FPTRATOD*

!         EXPLICIT ARGUMENTS
!         --------------------
!          INPUT:
!           KC:          1: one call of DIWRGRFP+DISGRIDFP per field
!           KFLDS:       number of fields to be transposed.
!           PFIELD_ARR:  array with departure geometry DM-distribution.
!          OUTPUT:
!           PFIELD_DEP:  array with arrival geometry DM-distribution.

!         IMPLICIT ARGUMENTS
!         --------------------
!         See lower #include.

!     METHOD.
!     -------
!        SEE DOCUMENTATION

!     EXTERNALS.
!     ----------

!     REFERENCE.
!     ----------
!        ECMWF Research Department documentation of the IFS
!        Documentation about FULL-POS.

!     AUTHOR.
```

```
!     -------
!        K. Yessad (CNRM/GMAP)
!        ORIGINAL : 21-Mar-2007

!     MODIFICATIONS.
!     --------------
!        K. Yessad (June 2007): adaptations for demonstration
!     End modifications
!     ------------------------------------------------------------------

USE PARKIND1  ,ONLY : JPIM     ,JPRB
USE YOMHOOK   ,ONLY : LHOOK,   DR_HOOK

USE YOMLUN    ,ONLY : NULOUT
USE YOMFPG    , ONLY : NFPRGPG   ,NFPRGPL   ,NFPRGPLX ,NFPRGPL_DEP,NFPRGPLX_DEP
USE YOMCT0    , ONLY : NPROC
USE YOMMP     , ONLY : MYPROC
USE YOMTAG    , ONLY : MTAGDISTGP          ,MTAGDISTGP_DEP
USE YOMFPGIND, ONLY : NFPRGPIND,NFPRGPNUM,NFPRGPIND_DEP,NFPRGPNUM_DEP

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

IMPLICIT NONE

INTEGER(KIND=JPIM),INTENT(IN) :: KC
INTEGER(KIND=JPIM),INTENT(IN) :: KFLDS
REAL(KIND=JPRB),INTENT(IN)    :: PFIELD_ARR(NFPRGPL,KFLDS)
REAL(KIND=JPRB),INTENT(OUT)   :: PFIELD_DEP(NFPRGPL_DEP,KFLDS)

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

REAL(KIND=JPRB) :: ZFIELDG(NFPRGPG,1+(KC-1)*(KFLDS-1))
INTEGER(KIND=JPIM) :: IAFPIO(KFLDS)
LOGICAL :: LLMASK(KFLDS)

INTEGER(KIND=JPIM) :: ISTRIN, JFIELD
INTEGER(KIND=JPIM) :: INFG, INFL, INFD(NPROC), IFLDOFF(NPROC)
CHARACTER(LEN=14)  :: CLDIAG
REAL(KIND=JPRB) :: ZHOOK_HANDLE
```

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

#include "disgridfp.intfb.h"
#include "diwrgrfp.intfb.h"

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

IF (LHOOK) CALL DR_HOOK('FPTRATOD',0,ZHOOK_HANDLE)

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

!*       1. PRELIMINARY CALCULATIONS
!           ------------------------

ISTRIN=NPROC
CLDIAG='CASE NOT CODED'

!*       2. READ OR COMPUTE OUTPUT CLIMATOLOGY
!           ---------------------------------

IF (KFLDS > 0) THEN

  !      2.1 Calculation of IAFPIO and LLMASK

  IAFPIO(:) = -999
  LLMASK(:) = .FALSE.
  DO JFIELD=1,KFLDS
    IAFPIO(JFIELD) = MOD(JFIELD-1,ISTRIN)+1
    LLMASK(JFIELD) = MYPROC /= IAFPIO(JFIELD)
  ENDDO

  !      2.2 Processor communications

  IF (NPROC == 1) THEN
    ! Data transfers
    PFIELD_DEP(:,:)=PFIELD_ARR(:,:)
  ELSE

    IF (KC == 1) THEN
```

```
      ! One call of DIWRGRFP+DISGRIDFP per field.
      INFG=1
      IFLDOFF(1:NPROC)=0
      DO JFIELD=1,KFLDS
        ! INFL is 1 when the current proc collects the DM-global array ZFIELDG,
        !  0 otherwise
        ! (if 0, DISGRIDFP only receives data, DIWRGRFP only sends data).
        IF (.NOT.LLMASK(JFIELD)) THEN
          INFL=1
        ELSE
          INFL=0
        ENDIF
        ! INFD is 1 for the proc which collects the DM-global array ZFIELDG,
        !  0 otherwise.
        INFD(1:NPROC)=0
        INFD(IAFPIO(JFIELD))=1
        CALL DIWRGRFP(MTAGDISTGP,INFG,NFPRGPG,NFPRGPL,NFPRGPLX,INFL, &
         & NFPRGPIND,NFPRGPNUM,INFD,IFLDOFF,PFIELD_ARR(1,JFIELD), &
         & ZFIELDG(1,1))
        CALL DISGRIDFP(MTAGDISTGP_DEP,INFG,NFPRGPG,NFPRGPL_DEP,NFPRGPLX_DEP,INFL, &
         & NFPRGPIND_DEP,NFPRGPNUM_DEP,INFD,IFLDOFF,ZFIELDG(1,1),&
         & PFIELD_DEP(1,JFIELD))
      ENDDO

    ELSE

      WRITE(NULOUT,'(1X,A)') CLDIAG

    ENDIF

  ENDIF
ENDIF

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

IF (LHOOK) CALL DR_HOOK('FPTRATOD',1,ZHOOK_HANDLE)
END SUBROUTINE FPTRATOD
```

SPECIFICATIONS : DOCUMENTATION.

External documentation :

- Separate scientific documentation, technical documentation and user's guide.

Internal documentation :

- Header comments stating briefly the purpose of the module, the author, references to external documentation, list of modifications, description of the dummy arguments.
- Section comments.
- Supplementary comments : should help reading the code, but the number of them should remain reasonable.

SPECIFICATIONS : DOCUMENTATION.

External documentation :

- Separate scientific documentation, technical documentation and user's guide.

Internal documentation :

- Header comments stating briefly the purpose of the module, the author, references to external documentation, list of modifications, description of the dummy arguments.
- Section comments.
- Supplementary comments : should help reading the code, but the number of them should remain reasonable.

SPECIFICATIONS : DOCUMENTATION.

External documentation :

- Separate scientific documentation, technical documentation and user's guide.

Internal documentation :

- Header comments stating briefly the purpose of the module, the author, references to external documentation, list of modifications, description of the dummy arguments.
- Section comments.
- Supplementary comments : should help reading the code, but the number of them should remain reasonable.

SPECIFICATIONS : DOCUMENTATION.

External documentation :

- Separate scientific documentation, technical documentation and user's guide.

Internal documentation :

- Header comments stating briefly the purpose of the module, the author, references to external documentation, list of modifications, description of the dummy arguments.
- Section comments.
- Supplementary comments : should help reading the code, but the number of them should remain reasonable.

SPECIFICATIONS : DOCUMENTATION.

External documentation :

- Separate scientific documentation, technical documentation and user's guide.

Internal documentation :

- Header comments stating briefly the purpose of the module, the author, references to external documentation, list of modifications, description of the dummy arguments.
- Section comments.
- Supplementary comments : should help reading the code, but the number of them should remain reasonable.

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Typewriting and basic layout :

- Executable lines in upper case characters, comments in lower case characters. No mix lower/upper case characters in executable lines !
- FORTRAN 90 free format.
- Not indented lines start at column 1.
- Ending statement : END SUBROUTINE SUPROGX
- Tabulations prohibited !
- One statement per line, no more !
- No more than 300 executable statements. No more than 2000 lines.
- Comments in English.
- Blank lines empty (no " !" in the first column).
- No alternate returns.
- Abnormal termination : CALL ABOR1(' SUPROGX : ... ')

DESIGN.

Declaring variables :

- Variable declaration modules : all declared variables should be commented ; 1 declaration per line, no more. Statement SAVE is compulsory.
- IMPLICIT NONE statement is mandatory !
- Avoid hard coded variables (such as CALL POSNAM(4,'NAMCT0'), or WRITE(6,*)). Never write on 0, 6 or 20 but write on NULERR or NULOUT. For namelist reading never read on 4, read on NULNAM.
- Dummy arguments in direct code : declare first integer dimension input variables, then other input variables, then input-output variables, then output variables.
- Dummy arguments in TL+AD code : same order as in the direct code for non-trajectory arguments, trajectory arguments (name ending by 5) are declared at the end.
- Dummy arguments : presentation, comments and declaration => same order !
- No more than 9 dummy arguments if possible.

DESIGN.

Declaring variables :

- Variable declaration modules : all declared variables should be commented ; 1 declaration per line, no more. Statement SAVE is compulsory.
- IMPLICIT NONE statement is mandatory !
- Avoid hard coded variables (such as CALL POSNAM(4,'NAMCT0'), or WRITE(6,*)). Never write on 0, 6 or 20 but write on NULERR or NULOUT. For namelist reading never read on 4, read on NULNAM.
- Dummy arguments in direct code : declare first integer dimension input variables, then other input variables, then input-output variables, then output variables.
- Dummy arguments in TL+AD code : same order as in the direct code for non-trajectory arguments, trajectory arguments (name ending by 5) are declared at the end.
- Dummy arguments : presentation, comments and declaration => same order !
- No more than 9 dummy arguments if possible.

DESIGN.

Declaring variables :

- Variable declaration modules : all declared variables should be commented ; 1 declaration per line, no more. Statement SAVE is compulsory.
- IMPLICIT NONE statement is mandatory !
- Avoid hard coded variables (such as CALL POSNAM(4,'NAMCT0'), or WRITE(6,*)). Never write on 0, 6 or 20 but write on NULERR or NULOUT. For namelist reading never read on 4, read on NULNAM.
- Dummy arguments in direct code : declare first integer dimension input variables, then other input variables, then input-output variables, then output variables.
- Dummy arguments in TL+AD code : same order as in the direct code for non-trajectory arguments, trajectory arguments (name ending by 5) are declared at the end.
- Dummy arguments : presentation, comments and declaration => same order !
- No more than 9 dummy arguments if possible.

DESIGN.

Declaring variables :

- Variable declaration modules : all declared variables should be commented ; 1 declaration per line, no more. Statement SAVE is compulsory.
- IMPLICIT NONE statement is mandatory !
- Avoid hard coded variables (such as CALL POSNAM(4,'NAMCT0'), or WRITE(6,*)). Never write on 0, 6 or 20 but write on NULERR or NULOUT. For namelist reading never read on 4, read on NULNAM.
- Dummy arguments in direct code : declare first integer dimension input variables, then other input variables, then input-output variables, then output variables.
- Dummy arguments in TL+AD code : same order as in the direct code for non-trajectory arguments, trajectory arguments (name ending by 5) are declared at the end.
- Dummy arguments : presentation, comments and declaration => same order !
- No more than 9 dummy arguments if possible.

DESIGN.

Declaring variables :

- Variable declaration modules : all declared variables should be commented ; 1 declaration per line, no more. Statement SAVE is compulsory.
- IMPLICIT NONE statement is mandatory !
- Avoid hard coded variables (such as CALL POSNAM(4,'NAMCT0'), or WRITE(6,*)). Never write on 0, 6 or 20 but write on NULERR or NULOUT. For namelist reading never read on 4, read on NULNAM.
- Dummy arguments in direct code : declare first integer dimension input variables, then other input variables, then input-output variables, then output variables.
- Dummy arguments in TL+AD code : same order as in the direct code for non-trajectory arguments, trajectory arguments (name ending by 5) are declared at the end.
- Dummy arguments : presentation, comments and declaration => same order !
- No more than 9 dummy arguments if possible.

DESIGN.

Declaring variables :

- Variable declaration modules : all declared variables should be commented ; 1 declaration per line, no more. Statement SAVE is compulsory.
- IMPLICIT NONE statement is mandatory !
- Avoid hard coded variables (such as CALL POSNAM(4,'NAMCT0'), or WRITE(6,*)). Never write on 0, 6 or 20 but write on NULERR or NULOUT. For namelist reading never read on 4, read on NULNAM.
- Dummy arguments in direct code : declare first integer dimension input variables, then other input variables, then input-output variables, then output variables.
- Dummy arguments in TL+AD code : same order as in the direct code for non-trajectory arguments, trajectory arguments (name ending by 5) are declared at the end.
- Dummy arguments : presentation, comments and declaration => same order !
- No more than 9 dummy arguments if possible.

DESIGN.

Declaring variables (cont'd) :

- Statement DIMENSION is prohibited !
- " : :" is mandatory between the type+attribute and the name of the declared variable.
- Unused variables (USE in a module, variable declaration) should not appear.
- Declare with an explicit kind (ex : JPIM, JPRB).
- Order : use module variables, declare dummy arguments, declare local variables.
- SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) : commas at the end when breaking lines.
- Callee SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) in caller : dummy arguments are in the same order even for optional arguments with an identifier.

Declaring variables (cont'd) :

- Statement DIMENSION is prohibited !
- " : :" is mandatory between the type+attribute and the name of the declared variable.
- Unused variables (USE in a module, variable declaration) should not appear.
- Declare with an explicit kind (ex : JPIM, JPRB).
- Order : use module variables, declare dummy arguments, declare local variables.
- SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) : commas at the end when breaking lines.
- Callee SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) in caller : dummy arguments are in the same order even for optional arguments with an identifier.

Declaring variables (cont'd) :

- Statement DIMENSION is prohibited !
- " : :" is mandatory between the type+attribute and the name of the declared variable.
- Unused variables (USE in a module, variable declaration) should not appear.
- Declare with an explicit kind (ex : JPIM, JPRB).
- Order : use module variables, declare dummy arguments, declare local variables.
- SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) : commas at the end when breaking lines.
- Callee SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) in caller : dummy arguments are in the same order even for optional arguments with an identifier.

Declaring variables (cont'd) :

- Statement DIMENSION is prohibited !
- " : :" is mandatory between the type+attribute and the name of the declared variable.
- Unused variables (USE in a module, variable declaration) should not appear.
- Declare with an explicit kind (ex : JPIM, JPRB).
- Order : use module variables, declare dummy arguments, declare local variables.
- SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) : commas at the end when breaking lines.
- Callee SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) in caller : dummy arguments are in the same order even for optional arguments with an identifier.

Declaring variables (cont'd) :

- Statement DIMENSION is prohibited !
- " : :" is mandatory between the type+attribute and the name of the declared variable.
- Unused variables (USE in a module, variable declaration) should not appear.
- Declare with an explicit kind (ex : JPIM, JPRB).
- Order : use module variables, declare dummy arguments, declare local variables.
- SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) : commas at the end when breaking lines.
- Callee SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) in caller : dummy arguments are in the same order even for optional arguments with an identifier.

Declaring variables (cont'd) :

- Statement DIMENSION is prohibited !
- " : :" is mandatory between the type+attribute and the name of the declared variable.
- Unused variables (USE in a module, variable declaration) should not appear.
- Declare with an explicit kind (ex : JPIM, JPRB).
- Order : use module variables, declare dummy arguments, declare local variables.
- SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) : commas at the end when breaking lines.
- Callee SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) in caller : dummy arguments are in the same order even for optional arguments with an identifier.

Declaring variables (cont'd) :

- Statement DIMENSION is prohibited !
- " : :" is mandatory between the type+attribute and the name of the declared variable.
- Unused variables (USE in a module, variable declaration) should not appear.
- Declare with an explicit kind (ex : JPIM, JPRB).
- Order : use module variables, declare dummy arguments, declare local variables.
- SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) : commas at the end when breaking lines.
- Callee SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) in caller : dummy arguments are in the same order even for optional arguments with an identifier.

Declaring variables (cont'd) :

- Statement DIMENSION is prohibited !
- " : :" is mandatory between the type+attribute and the name of the declared variable.
- Unused variables (USE in a module, variable declaration) should not appear.
- Declare with an explicit kind (ex : JPIM, JPRB).
- Order : use module variables, declare dummy arguments, declare local variables.
- SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) : commas at the end when breaking lines.
- Callee SUBROUTINE SUPROGX(...) and CALL SUPROGX(...) in caller : dummy arguments are in the same order even for optional arguments with an identifier.

DESIGN (CONT'D).

Loops, conditional blocks, linebreaking :

- DO loops :
  - A loop starts by DO, ends by ENDDO. Write ENDDO and not END DO.
  - Array syntax only for simple operations (set to a value, simple memory transfer).
- IF conditions :
  - Use the SELECT CASE statement when possible, rather than IF/ELSEIF/ELSE/ENDIF. Write ELSEIF, not ELSE IF. Write ENDIF, not END IF.
  - For a sequence of conditions in the same IF/ENDIF block, conditions should be exclusive.
  - Maximum 3 levels of conditional blocks nesting.
  - In comparison operators, use ==, /=, <, >, <=, >=, not .EQ., .NE., .LT., .GT., .LE., .GE. .
- Indentations :
  - Two blank spaces indentation for DO or DO WHILE loops and conditional blocks.
  - One blank space indentation for continuation lines.

DESIGN (CONT'D).

Loops, conditional blocks, linebreaking :

- DO loops :
  - A loop starts by DO, ends by ENDDO. Write ENDDO and not END DO.
  - Array syntax only for simple operations (set to a value, simple memory transfer).
- IF conditions :
  - Use the SELECT CASE statement when possible, rather than IF/ELSEIF/ELSE/ENDIF. Write ELSEIF, not ELSE IF. Write ENDIF, not END IF.
  - For a sequence of conditions in the same IF/ENDIF block, conditions should be exclusive.
  - Maximum 3 levels of conditional blocks nesting.
  - In comparison operators, use ==, /=, <, >, <=, >=, not .EQ., .NE., .LT., .GT., .LE., .GE. .
- Indentations :
  - Two blank spaces indentation for DO or DO WHILE loops and conditional blocks.
  - One blank space indentation for continuation lines.

DESIGN (CONT'D).

Loops, conditional blocks, linebreaking :

- DO loops :
  - A loop starts by DO, ends by ENDDO. Write ENDDO and not END DO.
  - Array syntax only for simple operations (set to a value, simple memory transfer).
- IF conditions :
  - Use the SELECT CASE statement when possible, rather than IF/ELSEIF/ELSE/ENDIF. Write ELSEIF, not ELSE IF. Write ENDIF, not END IF.
  - For a sequence of conditions in the same IF/ENDIF block, conditions should be exclusive.
  - Maximum 3 levels of conditional blocks nesting.
  - In comparison operators, use $==$, $/=$, $<$, $>$, $<=$, $>=$, not .EQ., .NE., .LT., .GT., .LE., .GE. .
- Indentations :
  - Two blank spaces indentation for DO or DO WHILE loops and conditional blocks.
  - One blank space indentation for continuation lines.

DESIGN (CONT'D).

Loops, conditional blocks, linebreaking :

- DO loops :
  - A loop starts by DO, ends by ENDDO. Write ENDDO and not END DO.
  - Array syntax only for simple operations (set to a value, simple memory transfer).
- IF conditions :
  - Use the SELECT CASE statement when possible, rather than IF/ELSEIF/ELSE/ENDIF. Write ELSEIF, not ELSE IF. Write ENDIF, not END IF.
  - For a sequence of conditions in the same IF/ENDIF block, conditions should be exclusive.
  - Maximum 3 levels of conditional blocks nesting.
  - In comparison operators, use $==$, $/=$, $<$, $>$, $<=$, $>=$, not .EQ., .NE., .LT., .GT., .LE., .GE. .
- Indentations :
  - Two blank spaces indentation for DO or DO WHILE loops and conditional blocks.
  - One blank space indentation for continuation lines.

BANNED FEATURES.

Avoid the following statements :

- COMMON (use MODULE instead).
- COMPLEX, DOUBLE PRECISION.
- CONTINUE.
- DIMENSION.
- ENTRY.
- EQUIVALENCE.
- FORMAT statement.
- GO TO.
- STOP.
- DATA.
- PRINT * (use WRITE(NULOUT,[format]) or WRITE(NULERR,[format]) instead).
- CHARACTER*n (use CHARACTER(LEN=n) instead).
- Declaration with implicit size (REAL(KIND=JPRB) : : A(*)).

BANNED FEATURES.

Avoid the following statements :

- COMMON (use MODULE instead).
- COMPLEX, DOUBLE PRECISION.
- CONTINUE.
- DIMENSION.
- ENTRY.
- EQUIVALENCE.
- FORMAT statement.
- GO TO.
- STOP.
- DATA.
- PRINT * (use WRITE(NULOUT,[format]) or WRITE(NULERR,[format]) instead).
- CHARACTER*n (use CHARACTER(LEN=n) instead).
- Declaration with implicit size (REAL(KIND=JPRB) : : A(*)).

BANNED FEATURES.

Avoid the following statements :

- COMMON (use MODULE instead).
- COMPLEX, DOUBLE PRECISION.
- CONTINUE.
- DIMENSION.
- ENTRY.
- EQUIVALENCE.
- FORMAT statement.
- GO TO.
- STOP.
- DATA.
- PRINT * (use WRITE(NULOUT,[format]) or WRITE(NULERR,[format]) instead).
- CHARACTER*n (use CHARACTER(LEN=n) instead).
- Declaration with implicit size (REAL(KIND=JPRB) : : A(*)).

NAMING CONVENTIONS FOR VARIABLES.

- Use the DOCTOR norm for naming variables (projects MPA ,MSE, SURFEX may use the MESO-NH DOCTOR).
- The root of a variable name should be meaningful for English-speakers.
- Ensure variable naming consistency for a same topic (ex : for land-sea-mask use only root LSM).
- Variables which have a DM-local and a DM-global version (local/global relative to the memory distribution) : name ends by L for local version, by G for global version (ex : NDGLL/NDGLG).
- Variables should not have the same name of a routine (for example intrinsic routine) which may be used in the same subroutine. For example a variable should not be called ISMAX, ISMIN, MINVAL.
- Avoid if possible more than 9 letters for a variable name.

NAMING CONVENTIONS FOR VARIABLES.

- Use the DOCTOR norm for naming variables (projects MPA ,MSE, SURFEX may use the MESO-NH DOCTOR).
- The root of a variable name should be meaningful for English-speakers.
- Ensure variable naming consistency for a same topic (ex : for land-sea-mask use only root LSM).
- Variables which have a DM-local and a DM-global version (local/global relative to the memory distribution) : name ends by L for local version, by G for global version (ex : NDGLL/NDGLG).
- Variables should not have the same name of a routine (for example intrinsic routine) which may be used in the same subroutine. For example a variable should not be called ISMAX, ISMIN, MINVAL.
- Avoid if possible more than 9 letters for a variable name.

NAMING CONVENTIONS FOR VARIABLES.

- Use the DOCTOR norm for naming variables (projects MPA ,MSE, SURFEX may use the MESO-NH DOCTOR).
- The root of a variable name should be meaningful for English-speakers.
- Ensure variable naming consistency for a same topic (ex : for land-sea-mask use only root LSM).
- Variables which have a DM-local and a DM-global version (local/global relative to the memory distribution) : name ends by L for local version, by G for global version (ex : NDGLL/NDGLG).
- Variables should not have the same name of a routine (for example intrinsic routine) which may be used in the same subroutine. For example a variable should not be called ISMAX, ISMIN, MINVAL.
- Avoid if possible more than 9 letters for a variable name.

NAMING CONVENTIONS FOR VARIABLES.

- Use the DOCTOR norm for naming variables (projects MPA ,MSE, SURFEX may use the MESO-NH DOCTOR).
- The root of a variable name should be meaningful for English-speakers.
- Ensure variable naming consistency for a same topic (ex : for land-sea-mask use only root LSM).
- Variables which have a DM-local and a DM-global version (local/global relative to the memory distribution) : name ends by L for local version, by G for global version (ex : NDGLL/NDGLG).
- Variables should not have the same name of a routine (for example intrinsic routine) which may be used in the same subroutine. For example a variable should not be called ISMAX, ISMIN, MINVAL.
- Avoid if possible more than 9 letters for a variable name.

NAMING CONVENTIONS FOR VARIABLES.

- Use the DOCTOR norm for naming variables (projects MPA ,MSE, SURFEX may use the MESO-NH DOCTOR).
- The root of a variable name should be meaningful for English-speakers.
- Ensure variable naming consistency for a same topic (ex : for land-sea-mask use only root LSM).
- Variables which have a DM-local and a DM-global version (local/global relative to the memory distribution) : name ends by L for local version, by G for global version (ex : NDGLL/NDGLG).
- Variables should not have the same name of a routine (for example intrinsic routine) which may be used in the same subroutine. For example a variable should not be called ISMAX, ISMIN, MINVAL.
- Avoid if possible more than 9 letters for a variable name.

NAMING CONVENTIONS FOR VARIABLES.

- Use the DOCTOR norm for naming variables (projects MPA ,MSE, SURFEX may use the MESO-NH DOCTOR).
- The root of a variable name should be meaningful for English-speakers.
- Ensure variable naming consistency for a same topic (ex : for land-sea-mask use only root LSM).
- Variables which have a DM-local and a DM-global version (local/global relative to the memory distribution) : name ends by L for local version, by G for global version (ex : NDGLL/NDGLG).
- Variables should not have the same name of a routine (for example intrinsic routine) which may be used in the same subroutine. For example a variable should not be called ISMAX, ISMIN, MINVAL.
- Avoid if possible more than 9 letters for a variable name.

NAMING CONVENTIONS FOR VARIABLES.

- Use the DOCTOR norm for naming variables (projects MPA ,MSE, SURFEX may use the MESO-NH DOCTOR).
- The root of a variable name should be meaningful for English-speakers.
- Ensure variable naming consistency for a same topic (ex : for land-sea-mask use only root LSM).
- Variables which have a DM-local and a DM-global version (local/global relative to the memory distribution) : name ends by L for local version, by G for global version (ex : NDGLL/NDGLG).
- Variables should not have the same name of a routine (for example intrinsic routine) which may be used in the same subroutine. For example a variable should not be called ISMAX, ISMIN, MINVAL.
- Avoid if possible more than 9 letters for a variable name.

The ARPEGE DOCTOR norm writes:

Status / 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
Derived type	Y but not (YD,YL,YP)	YD	YL	/	YP

The MESO-NH DOCTOR norm writes:

Status / Type	Variable in data module	Dummy argument	Local variable	Loop control	Any parameter	Save
INTEGER	N	K	I but not IS	J but not JP	JP	IS
REAL	X	P but not PP	Z but not ZS	/	PP	ZS
LOGICAL	L but not LP	O	G but not GS	/	LP	GS
CHARACTER	C	H	Y but not (YS,YP)	/	YP	YS
Derived type	T but not (TP,TS,TZ)	TP	TZ	/	/	TS

NAMING CONVENTIONS FOR DECLARATION MODULES.

Prefixations

The list of prefixations can be summarized as follows :

- PAR (PER for ALADIN) : parameter declaration and set-up.
- PTR : pointer declaration.
- QA : variables used in CANARI.
- QAPA : parameter variables used in CANARI.
- YEM : variables used in ALADIN only.
- YHL : variables used in the HIRLAM physics only.
- YOE : variables used in the ECMWF physics only.
- YOP : variables used in the simplified ECMWF physics only.
- YOMFP : specific FULL-POS variables.
- YOS_ (project SUR) : specific ECMWF surface scheme variables.
- TPM_ (project TFL) : spectral transforms variables (and type definitions).
- TPMALD_ (project TAL) : spectral transforms variables (and type definitions), for ALADIN only.
- YOM : other variables.
- TYPE_ : type definition.

NAMING CONVENTIONS FOR DECLARATION MODULES.

Prefixations

The list of prefixations can be summarized as follows :

- PAR (PER for ALADIN) : parameter declaration and set-up.
- PTR : pointer declaration.
- QA : variables used in CANARI.
- QAPA : parameter variables used in CANARI.
- YEM : variables used in ALADIN only.
- YHL : variables used in the HIRLAM physics only.
- YOE : variables used in the ECMWF physics only.
- YOP : variables used in the simplified ECMWF physics only.
- YOMFP : specific FULL-POS variables.
- YOS_ (project SUR) : specific ECMWF surface scheme variables.
- TPM_ (project TFL) : spectral transforms variables (and type definitions).
- TPMALD_ (project TAL) : spectral transforms variables (and type definitions), for ALADIN only.
- YOM : other variables.
- TYPE_ : type definition.

NAMING CONVENTIONS FOR DECLARATION MODULES.

Prefixations

The list of prefixations can be summarized as follows :

- PAR (PER for ALADIN) : parameter declaration and set-up.
- PTR : pointer declaration.
- QA : variables used in CANARI.
- QAPA : parameter variables used in CANARI.
- YEM : variables used in ALADIN only.
- YHL : variables used in the HIRLAM physics only.
- YOE : variables used in the ECMWF physics only.
- YOP : variables used in the simplified ECMWF physics only.
- YOMFP : specific FULL-POS variables.
- YOS_ (project SUR) : specific ECMWF surface scheme variables.
- TPM_ (project TFL) : spectral transforms variables (and type definitions).
- TPMALD_ (project TAL) : spectral transforms variables (and type definitions), for ALADIN only.
- YOM : other variables.
- TYPE_ : type definition.

NAMING CONVENTIONS FOR DECLARATION MODULES.

Prefixations

The list of prefixations can be summarized as follows :

- PAR (PER for ALADIN) : parameter declaration and set-up.
- PTR : pointer declaration.
- QA : variables used in CANARI.
- QAPA : parameter variables used in CANARI.
- YEM : variables used in ALADIN only.
- YHL : variables used in the HIRLAM physics only.
- YOE : variables used in the ECMWF physics only.
- YOP : variables used in the simplified ECMWF physics only.
- YOMFP : specific FULL-POS variables.
- YOS_ (project SUR) : specific ECMWF surface scheme variables.
- TPM_ (project TFL) : spectral transforms variables (and type definitions).
- TPMALD_ (project TAL) : spectral transforms variables (and type definitions), for ALADIN only.
- YOM : other variables.
- TYPE_ : type definition.

NAMING CONVENTIONS FOR DECLARATION MODULES.

Prefixations

The list of prefixations can be summarized as follows :

- PAR (PER for ALADIN) : parameter declaration and set-up.
- PTR : pointer declaration.
- QA : variables used in CANARI.
- QAPA : parameter variables used in CANARI.
- YEM : variables used in ALADIN only.
- YHL : variables used in the HIRLAM physics only.
- YOE : variables used in the ECMWF physics only.
- YOP : variables used in the simplified ECMWF physics only.
- YOMFP : specific FULL-POS variables.
- YOS_ (project SUR) : specific ECMWF surface scheme variables.
- TPM_ (project TFL) : spectral transforms variables (and type definitions).
- TPMALD_ (project TAL) : spectral transforms variables (and type definitions), for ALADIN only.
- YOM : other variables.
- TYPE_ : type definition.

NAMING CONVENTIONS FOR DECLARATION MODULES.

Prefixations

The list of prefixations can be summarized as follows :

- PAR (PER for ALADIN) : parameter declaration and set-up.
- PTR : pointer declaration.
- QA : variables used in CANARI.
- QAPA : parameter variables used in CANARI.
- YEM : variables used in ALADIN only.
- YHL : variables used in the HIRLAM physics only.
- YOE : variables used in the ECMWF physics only.
- YOP : variables used in the simplified ECMWF physics only.
- YOMFP : specific FULL-POS variables.
- YOS_ (project SUR) : specific ECMWF surface scheme variables.
- TPM_ (project TFL) : spectral transforms variables (and type definitions).
- TPMALD_ (project TAL) : spectral transforms variables (and type definitions), for ALADIN only.
- YOM : other variables.
- TYPE_ : type definition.

NAMING CONVENTIONS FOR DECLARATION MODULES.

Prefixations

The list of prefixations can be summarized as follows :

- PAR (PER for ALADIN) : parameter declaration and set-up.
- PTR : pointer declaration.
- QA : variables used in CANARI.
- QAPA : parameter variables used in CANARI.
- YEM : variables used in ALADIN only.
- YHL : variables used in the HIRLAM physics only.
- YOE : variables used in the ECMWF physics only.
- YOP : variables used in the simplified ECMWF physics only.
- YOMFP : specific FULL-POS variables.
- YOS_ (project SUR) : specific ECMWF surface scheme variables.
- TPM_ (project TFL) : spectral transforms variables (and type definitions).
- TPMALD_ (project TAL) : spectral transforms variables (and type definitions), for ALADIN only.
- YOM : other variables.
- TYPE_ : type definition.

NAMING CONVENTIONS FOR DECLARATION MODULES.

Prefixations

The list of prefixations can be summarized as follows :

- PAR (PER for ALADIN) : parameter declaration and set-up.
- PTR : pointer declaration.
- QA : variables used in CANARI.
- QAPA : parameter variables used in CANARI.
- YEM : variables used in ALADIN only.
- YHL : variables used in the HIRLAM physics only.
- YOE : variables used in the ECMWF physics only.
- YOP : variables used in the simplified ECMWF physics only.
- YOMFP : specific FULL-POS variables.
- YOS_ (project SUR) : specific ECMWF surface scheme variables.
- TPM_ (project TFL) : spectral transforms variables (and type definitions).
- TPMALD_ (project TAL) : spectral transforms variables (and type definitions), for ALADIN only.
- YOM : other variables.
- TYPE_ : type definition.

NAMING CONVENTIONS FOR NAMELISTS.

Prefixations

The list of prefixations can be summarized as follows :

- NAC (sometimes NAI, NAL) : namelists for CANARI.
- NAE : namelists for the ECMWF physics.
- NAP : namelists for simplified physics.
- NEM : specific ALADIN namelists.
- NAM : other namelists.

NAMING CONVENTIONS FOR NAMELISTS.

Prefixations

The list of prefixations can be summarized as follows :

- NAC (sometimes NAI, NAL) : namelists for CANARI.
- NAE : namelists for the ECMWF physics.
- NAP : namelists for simplified physics.
- NEM : specific ALADIN namelists.
- NAM : other namelists.

NAMING CONVENTIONS FOR OTHER ROUTINES.

Prefixations

The list of prefixations is difficult to provide extensively : here are some examples :

- CA but not CAIN nor CALL : CANARI.
- DFI : DFI initialisation.
- DIS, DIWR, GATHER, BROADC : distributed memory communication.
- FA, LFI, LFA (in XRD) : ARPEGE, LFI, LFA files.
- FP : FULL-POS.
- GP : grid-point calculations (low-level routines in the organigramme computing some well identified meteorological variables).
- LA : semi-Lagrangian scheme (LAI for interpolators, ELA for ALADIN).
- MPL_ (in XRD) : MPL software for processor communication.
- PP (in pp_obs) : vertical interpolator.
- SI but not SIM, SIPC : semi-implicit scheme (SIE for ALADIN).
- SP : spectral calculations (ESP for ALADIN).
- SU (SUE for ALADIN), but not (SURF,SUERF) : set-up routines.
- TR (in arp/parallel) : transpositions.
- WR : file writing.

NAMING CONVENTIONS FOR OTHER ROUTINES.

Prefixations

The list of prefixations is difficult to provide extensively : here are some examples :

- CA but not CAIN nor CALL : CANARI.
- DFI : DFI initialisation.
- DIS, DIWR, GATHER, BROADC : distributed memory communication.
- FA, LFI, LFA (in XRD) : ARPEGE, LFI, LFA files.
- FP : FULL-POS.
- GP : grid-point calculations (low-level routines in the organigramme computing some well identified meteorological variables).
- LA : semi-Lagrangian scheme (LAI for interpolators, ELA for ALADIN).
- MPL_ (in XRD) : MPL software for processor communication.
- PP (in pp_obs) : vertical interpolator.
- SI but not SIM, SIPC : semi-implicit scheme (SIE for ALADIN).
- SP : spectral calculations (ESP for ALADIN).
- SU (SUE for ALADIN), but not (SURF,SUERF) : set-up routines.
- TR (in arp/parallel) : transpositions.
- WR : file writing.

NAMING CONVENTIONS (CONT'D).

Suffixations

The list of suffixations can be summarized as follows :

- Modules containing executable code must have a name ending by _mod.F90.
- Tangent linear routines must have a name ending by TL.
- Adjoint routines must have a name ending by AD.
- Trajectory routines (for configurations using TL or AD codes) must have a name ending by 5 (but not 15 which is a specific suffixation for FMR-15 radiation scheme).

NAMING CONVENTIONS (CONT'D).

Suffixations

The list of suffixations can be summarized as follows :

- Modules containing executable code must have a name ending by _mod.F90.
- Tangent linear routines must have a name ending by TL.
- Adjoint routines must have a name ending by AD.
- Trajectory routines (for configurations using TL or AD codes) must have a name ending by 5 (but not 15 which is a specific suffixation for FMR-15 radiation scheme).

NAMING CONVENTIONS (CONT'D).

Suffixations

The list of suffixations can be summarized as follows :

- Modules containing executable code must have a name ending by _mod.F90.
- Tangent linear routines must have a name ending by TL.
- Adjoint routines must have a name ending by AD.
- Trajectory routines (for configurations using TL or AD codes) must have a name ending by 5 (but not 15 which is a specific suffixation for FMR-15 radiation scheme).

NAMING CONVENTIONS (CONT'D).

Suffixations

The list of suffixations can be summarized as follows :

- Modules containing executable code must have a name ending by _mod.F90.
- Tangent linear routines must have a name ending by TL.
- Adjoint routines must have a name ending by AD.
- Trajectory routines (for configurations using TL or AD codes) must have a name ending by 5 (but not 15 which is a specific suffixation for FMR-15 radiation scheme).

OTHER RECOMMENDATIONS.

- Universal constants are in YOMCST and set-up in SUCST : they should not be redefined elsewhere (for number Pi one should use variable RPI, not 3.1415926).
- LECMWF allowed in set-up routines only.
- the key LECMWF should not be used to select the format file reading or writing, but the proper key LARPEGEF, LARPEGEF_TRAJHR, LARPEGEF_TRAJBG or LGRBOP (according to the topic) should be used instead.
- LELAM (limited area models) allowed in set-up and control routines only.
- LRPLANE (plane geometry) can be used anywhere ; (LELAM,LRPLANE)=(T,F) has a sense (toric model) but is not implemented.
- Avoid code duplication ; the same application should not be done by two different routines.

OTHER RECOMMENDATIONS.

- Universal constants are in YOMCST and set-up in SUCST : they should not be redefined elsewhere (for number Pi one should use variable RPI, not 3.1415926).
- LECMWF allowed in set-up routines only.
- the key LECMWF should not be used to select the format file reading or writing, but the proper key LARPEGEF, LARPEGEF_TRAJHR, LARPEGEF_TRAJBG or LGRBOP (according to the topic) should be used instead.
- LELAM (limited area models) allowed in set-up and control routines only.
- LRPLANE (plane geometry) can be used anywhere ; (LELAM,LRPLANE)=(T,F) has a sense (toric model) but is not implemented.
- Avoid code duplication ; the same application should not be done by two different routines.

OTHER RECOMMENDATIONS.

- Universal constants are in YOMCST and set-up in SUCST : they should not be redefined elsewhere (for number Pi one should use variable RPI, not 3.1415926).
- LECMWF allowed in set-up routines only.
- the key LECMWF should not be used to select the format file reading or writing, but the proper key LARPEGEF, LARPEGEF_TRAJHR, LARPEGEF_TRAJBG or LGRBOP (according to the topic) should be used instead.
- LELAM (limited area models) allowed in set-up and control routines only.
- LRPLANE (plane geometry) can be used anywhere ; (LELAM,LRPLANE)=(T,F) has a sense (toric model) but is not implemented.
- Avoid code duplication ; the same application should not be done by two different routines.

OTHER RECOMMENDATIONS.

- Universal constants are in YOMCST and set-up in SUCST : they should not be redefined elsewhere (for number Pi one should use variable RPI, not 3.1415926).
- LECMWF allowed in set-up routines only.
- the key LECMWF should not be used to select the format file reading or writing, but the proper key LARPEGEF, LARPEGEF_TRAJHR, LARPEGEF_TRAJBG or LGRBOP (according to the topic) should be used instead.
- LELAM (limited area models) allowed in set-up and control routines only.
- LRPLANE (plane geometry) can be used anywhere ; (LELAM,LRPLANE)=(T,F) has a sense (toric model) but is not implemented.
- Avoid code duplication ; the same application should not be done by two different routines.

OTHER RECOMMENDATIONS.

- Universal constants are in YOMCST and set-up in SUCST : they should not be redefined elsewhere (for number Pi one should use variable RPI, not 3.1415926).
- LECMWF allowed in set-up routines only.
- the key LECMWF should not be used to select the format file reading or writing, but the proper key LARPEGEF, LARPEGEF_TRAJHR, LARPEGEF_TRAJBG or LGRBOP (according to the topic) should be used instead.
- LELAM (limited area models) allowed in set-up and control routines only.
- LRPLANE (plane geometry) can be used anywhere ; (LELAM,LRPLANE)=(T,F) has a sense (toric model) but is not implemented.
- Avoid code duplication ; the same application should not be done by two different routines.

PRESENTATION NORMS FOR NAMELISTS.

- Namelists elements :
  - Should be in the alphabetical order.
  - All the existing elements should be referenced (NAC., NAI., NAL., NAE., NAP., NAM., NEM.).
  - Each referenced element appears only once.
  - No obsolete element should appear.
- Variables in each element :
  - In each element, there is one (and no more) variable per line.
  - Each referenced variable appears only once.
  - No obsolete variable should appear.
  - Fill arrays elements in the right order (for ex : filling element 1 then 2 then 3 is OK ; filling element 1 then 3 then 2 is not OK).
  - Each line containing a variable or a value should end by a comma.

PRESENTATION NORMS FOR NAMELISTS.

- Namelists elements :
  - Should be in the alphabetical order.
  - All the existing elements should be referenced (NAC., NAI., NAL., NAE., NAP., NAM., NEM.).
  - Each referenced element appears only once.
  - No obsolete element should appear.
- Variables in each element :
  - In each element, there is one (and no more) variable per line.
  - Each referenced variable appears only once.
  - No obsolete variable should appear.
  - Fill arrays elements in the right order (for ex : filling element 1 then 2 then 3 is OK ; filling element 1 then 3 then 2 is not OK).
  - Each line containing a variable or a value should end by a comma.

PRESENTATION NORMS FOR NAMELISTS.

- Namelists elements :
  - Should be in the alphabetical order.
  - All the existing elements should be referenced (NAC., NAI., NAL., NAE., NAP., NAM., NEM.).
  - Each referenced element appears only once.
  - No obsolete element should appear.
- Variables in each element :
  - In each element, there is one (and no more) variable per line.
  - Each referenced variable appears only once.
  - No obsolete variable should appear.
  - Fill arrays elements in the right order (for ex : filling element 1 then 2 then 3 is OK ; filling element 1 then 3 then 2 is not OK).
  - Each line containing a variable or a value should end by a comma.

PRESENTATION NORMS FOR NAMELISTS (CONT'D).

- Indentations and blank characters :
  - No blank character is allowed before or after the sign "=", and before the final comma ; for example one should write "NRADFR=-1,", but never "NRADFR =-1,", "NRADFR= -1," or "NRADFR=-1 ,".
  - Blank characters are allowed before the name of the variable, but all variables should be aligned. The current standard is one blank character before &NAM... and "slash", three blank characters before each variable.
- For logical variables, one always write .TRUE. or .FALSE. ; for example one should write "LREGETA=.FALSE.,", but never "LREGETA=.false.,", "LREGETA=.f.,", "LREGETA=.F.,", "LREGETA=F,".
- Only uppercase letters are allowed ; for example one should write "NRADFR=-1,", but never "nradfr=-1,".
- Normalization tools : xpnam, alignnamelist.

PRESENTATION NORMS FOR NAMELISTS (CONT'D).

- Indentations and blank characters :
  - No blank character is allowed before or after the sign "=", and before the final comma ; for example one should write "NRADFR=-1,", but never "NRADFR =-1,", "NRADFR= -1," or "NRADFR=-1 ,".
  - Blank characters are allowed before the name of the variable, but all variables should be aligned. The current standard is one blank character before &NAM... and "slash", three blank characters before each variable.
- For logical variables, one always write .TRUE. or .FALSE. ; for example one should write "LREGETA=.FALSE.,", but never "LREGETA=.false.,", "LREGETA=.f.,", "LREGETA=.F.,", "LREGETA=F,".
- Only uppercase letters are allowed ; for example one should write "NRADFR=-1,", but never "nradfr=-1,".
- Normalization tools : xpnam, alignnamelist.

PRESENTATION NORMS FOR NAMELISTS (CONT'D).

- Indentations and blank characters :
  - No blank character is allowed before or after the sign "=", and before the final comma ; for example one should write "NRADFR=-1,", but never "NRADFR =-1,", "NRADFR= -1," or "NRADFR=-1 ,".
  - Blank characters are allowed before the name of the variable, but all variables should be aligned. The current standard is one blank character before &NAM... and "slash", three blank characters before each variable.
- For logical variables, one always write .TRUE. or .FALSE. ; for example one should write "LREGETA=.FALSE.,", but never "LREGETA=.false.,", "LREGETA=.f.,", "LREGETA=.F.,", "LREGETA=F,".
- Only uppercase letters are allowed ; for example one should write "NRADFR=-1,", but never "nradfr=-1,".
- Normalization tools : xpnam, alignnamelist.

PRESENTATION NORMS FOR NAMELISTS (CONT'D).

- Indentations and blank characters :
  - No blank character is allowed before or after the sign "=", and before the final comma ; for example one should write "NRADFR=-1,", but never "NRADFR =-1,", "NRADFR= -1," or "NRADFR=-1 ,".
  - Blank characters are allowed before the name of the variable, but all variables should be aligned. The current standard is one blank character before &NAM... and "slash", three blank characters before each variable.
- For logical variables, one always write .TRUE. or .FALSE. ; for example one should write "LREGETA=.FALSE.,", but never "LREGETA=.false.,", "LREGETA=.f.,", "LREGETA=.F.,", "LREGETA=F,".
- Only uppercase letters are allowed ; for example one should write "NRADFR=-1,", but never "nradfr=-1,".
- Normalization tools : xpnam, alignnamelist.

THANK YOU / MERCI.