ALADIN PHASER'S GUIDE

By R. AJJAJI &J. BOUTAHAR (Maroc-Météo) & JF. GELEYN (Météo-France)

June 30, 1998, VERSION 1.1

Contents

INTRODUCTION

- The ALADIN code is following the ARPEGE one in its permanent evolution. This is due mainly to the technical evolution of calculating machines (CDC, CRAY 2, CRAY C90, FUJITSU, ...) which imposes a new adaptation of the code to the new machine properties. The evolution of the number of observations over the world dictates also a code development if one wants to take new observations into account. In addition to these two external constraints, code must contain the newest coding rules if one wants to have a permanent portable one, ... Code optimization as bugs correction, computation time and memory winning, ... is also another constraint. And of course, the evolution of the code is also imposed by the newdevelopments (A development usually involves another one !).
- The code can be modified for the above-mentioned purposes by a limited number of known developers or phasers to avoid complexity and misunderstanding consequences. Thus, ALADIN code can be touched only by GMAP team, ALADIN partners, some SCEM teams (COMPAS) and of course persons who come to GMAP for training.
- The software validation after modification is an absolute must. The phaser or developper must intensively validate its work. After the entire merge of modifications, all the system configurations must be tested and validated. And before using the new software for operational purpose, a double suite is strongly recommended.
- So a phaser must keep in mind that he has to respect all coding rules, to check all interactions with the remaining code at each step. He must also phase his scripts and namelists and keep himself informed about all what is being performed into the whole code at the same time.

I. CODING RULES:

There are several ways to write a program (or a software) for a given aim, and if all the methods make it possible to obtain correct results (files, listings, ...), they are not equivalent in general according to

other criterions.

So it is essential that a program (or a group of programs) should be:

- Well planned and designed.
- Well written.
- Sufficiently documented.
- Maintainable.
- Transportable.
- Efficient.
- Flexible.

Each module should begin with a set of principal comments. These should contain:

- A title.
- The Purpose of the routine.
- The interface details of input / output parameters, arguments, etc ... .
- The externals or other routines called.
- A reference to further documentation.
- The author and date with modifications details.

The code of the module should be split into sections and sub-sections. Each section should be separated from the previous by homogeneous numericallabels.

The type of a variable is indicated by the prefix of the variable names (cf : following tables)

PREFIX	TYPE
I,J,K,M,N	INTEGER
L	LOGICAL
C	CHARACTER
D	DOUBLE PRECISION
Y	COMPLEX
ALL OTHER	REAL

The following table defines the prefixesconventions (Clochard, J., 1998, Norme de codage "DOCTOR" pour le projet ARPEGE. Note de travail "ARPEGE" No. 4, and, Gibson, J.K., 1986, Standards for software development and maintenance,Technical memorandumNo. 120)

TYPE STATUS OR SCOPE	INTEGER	REAL	LOGICAL	CHARACTER	DOUBLE PRECISION	COMPLEX
MODULE OR COMMON	M, N	G,H,O Q To X	L but not LP,LD,LL	C but not CP,CD,CL	D but not DP,DD,DL	Y but not YP,YD,YL

DUMMY- ARGUMENT	K	P but not PP	LD	CD	DD	YD
LOCAL VARIABLES	I	Z	LL	CL	DL	YL
LOOP CONTROL	J but not JP	_	_	_	_	_
PARAMETER	JP	PP	LP	CP	DP	YP

Once a routine is written or modified, it should be reviewed. And the following should be considered:

- Does the source code satisfies the standards?
- Can the source code satisfy a standard ANSI Compiler?
- Is the code easy to understand?
- Is the code too complex?
- Is the interface to the routines simple and sufficient?
- Does the routine produce the required results?
- Can modification be made if required?
- Are error situations detected and correctly treated?
- Is the routine efficient?

The following example illustrates the rules above mentioned :

```
SUBROUTINE CNT2

!**** *CNT2* - Controls integration job at level 2.

! Purpose.

! --------

! Controls integration job at level 2.

!** Interface.

! ----------

! *CALL* *CNT2

! Explicit arguments :

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

! None

! Implicit arguments :

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

```
! None
! Method.
! -------
! See documentation
! Externals. SU2YOM - initialize level 2 commons
! ---------- CNT3 - controls direct integration on level 3
! Reference.
! ----------
! ECMWF Research Department documentation of the IFS
! Author.
! -------
! Mats Hamrud and Philippe Courtier *ECMWF*
! Modifications.
! --------------
! Original : 87-10-15
! Modified : 91-09-25 Jean-Noel THEPAUT. cleaning of TL and AD.
! Modified : 96-09-25 Florence Rabier. allocation for nconf=801
! -----------------------------------------------------------------
USE YOMCT0 , ONLY : NCONF ,LELAM
IMPLICIT LOGICAL(L)
! -----------------------------------------------------------------
!* 1. Initialization.
!* 1.1 Initialize YOMCT2.
CALL SU2YOM
!* 1.2 ALLOCATE EXTRA SPACE FOR TRAJECTORY.
IF(NCONF/100.EQ.1.OR.NCONF.EQ.801) THEN
CALL SUALLT
IF (LELAM) CALL SUELLT
```

```
ENDIF

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

!* 2. Call level 3 control routine.

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

CALL CNT3

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

RETURN

END SUBROUTINE CNT2

---------------------------------------------------------------------
-------------
```

II. HOW ALADIN CODE MAINTENANCE MUST BE DONE ?

II.1. GENERALITIES ABOUT IFS / ARPEGE / ALADIN

- IFS = ARPEGE & ALADIN dependent on ARPEGE
- Same application of the «IFMG» (Integration, Flexibility, Modularity, Generality) rule, same set of coding conventions, same type of scientific work but a different procedure of maintenance
- ALADIN specificities in geometry, coupling, spectral transforms and non-hydrostatic version.
- Something happening in ARPEGE always may affect ALADIN, sometimes in an unexpected manner!

II.2. THE MAINTENANCE OF ARPEGE/ALADIN SOFTWARE.

- A common Cycle ARPEGE-ALADIN (full number) every 6 months.
- Phasers meeting at the same frequency to create the new common cycle.
- Phasing <<at distance>> in some cases (E-mail exchanges).
- Little overlap of the crucial operational contingencies.
- But a lot of little «local» incompatibilities.

II.3. CODE MANAGEMENT (CLEAR CASE)

- ClearCase = toolbox used at Meteo-France to manage source code.
- Supervised by ClearCase Administrator: the <<gourous>>.
- One must create a new branch in ClearCase once its development is achieved.
- The merge is done by the <<gourous>> with phasers or developers collaboration.
- Easy way to learn ClearCase, (cf: Elkhatib, R., 1997, Easy ClearCase manual in 7 lessons and exercises).

II.4. NOTION OF CYCLES.

II.4.1 IN IFS/ARPEGE

- Cycle = Merge of new developments that have happened on each of the two sites Meteo/France and ECMWF (and inside each of them in the build-up).
- Identify (and remove or go around) incompatibilities between the different contributions, as well as «bugs in spe» due to the lack of knowledge of the other's plans (when possible)
- Validate as intensively as possible the code on the occasion of these special «rendez-vous».
- Build on the scientific innovation of Cycles and on their more intensive validation to increment the operational versions

II.4.2 TYPE OF CYCLES AND THEIR USE.

- Full Cycles, eg : CY18 , AL08, ...
- Interim Cycles (r & t), eg : CY18T1, CY18T2, CY18T3 , CY13R5, ...
- Operational Cycles, eg : CY18T1-AL08 , CY16T1-AL07, ...
- Bug-fixes, eg : CY18T1_bf1 , AL08_bf1, ...

II.4.3 SPECIFICITIES OF ALADIN CYCLES.

- Practical impossibility to follow the same strategy as for IFS/ARPEGE.
- Consistent choice of a non-integrated extension without interim cycles.
- Link with ARPEGE Cycles.
- Need to follow the ARPEGE basic evolution.
- ALADIN in ARPEGE code (for compatibility reasons).
- Anticipation of ALADIN's evolution in some ARPEGE Cycles.
- Problem of ALADIN back-phasing (cf: Annex C).
- The difficult balance between independence and a feasible maintenance.
- The important past and present evolutions.

II.5. IMPLICATIONS FOR THE ALADIN «AT DISTANCE» MAINTENANCE.

- It cannot be ensured like the IFS/ARPEGE one.
- It needs more planning and a better respect of calendars.
- A six month sleeping work is «dead» in this context.
- The need to incorporate at least the specificities of more and more deported operational applications will become a tough challenge.
- Meteo-France's partner NMSs should become far more aware of the importance of this question.

II.6. WORKSTATION VERSION OF ALADIN .

II.6.1. GENERALITIES.

- Such a version exists since the conversion to FORTRAN 90 and a specific work by SELAM colleagues in 95 allowed it.
- The workstation version differs from the main, supercomputer version by a few, localized statements.
- In future, it will be necessary to maintain the phasing between the workstation version and the main code evolution. Furthermore, scientific developments might be carried out on workstation, which

then shall be introduced in the general code. Therefore, the situation of the workstation version of ALADIN with respect to ALADIN might become closer to the one of ALADIN with respect to ARPEGE, with the production of new cycles delayed in time compared to the main ALADIN cycles. However there will be never an automatic path for reinjecting all workstation version's specificities into the main code.

II.6.2. SOME PRACTICAL ASPECTS.

- A good version of Fortran 90 is recommended.
- Localized adaptations of the auxiliary libraries.
- Specification of some machine properties in somexrd decks (lficom0.h,gsbyte_mf.f,xbyte.c).
- Put all namelist parameters LIOXXX to .FALSE. and the key LCRAYPVP to .FALSE..
- When running software use IEEE input files (use export_fa.sh to transformefiles from Cray format to IEEE one).

III. METHODOLOGY TO FOLLOW WHEN PHASING:

First it is the responsibility of ALADIN team to decide which ARPEGE cycle to phase with ALADIN and which components to report into ALADIN.

III.1. What is the composition of a new phased ALADIN cycle?

- Specific ALADIN modification (new pure ALADIN development (example: in coupling,...)).
- Modification coming from the new ARPEGE cycle.

III.2. Methodology.

III.2.1. Pure ALADIN Modifications:

The Phaser must take them under LELAM and/or LRPLANE keys.

III.2.2. Modifications coming from the new ARPEGE cycle

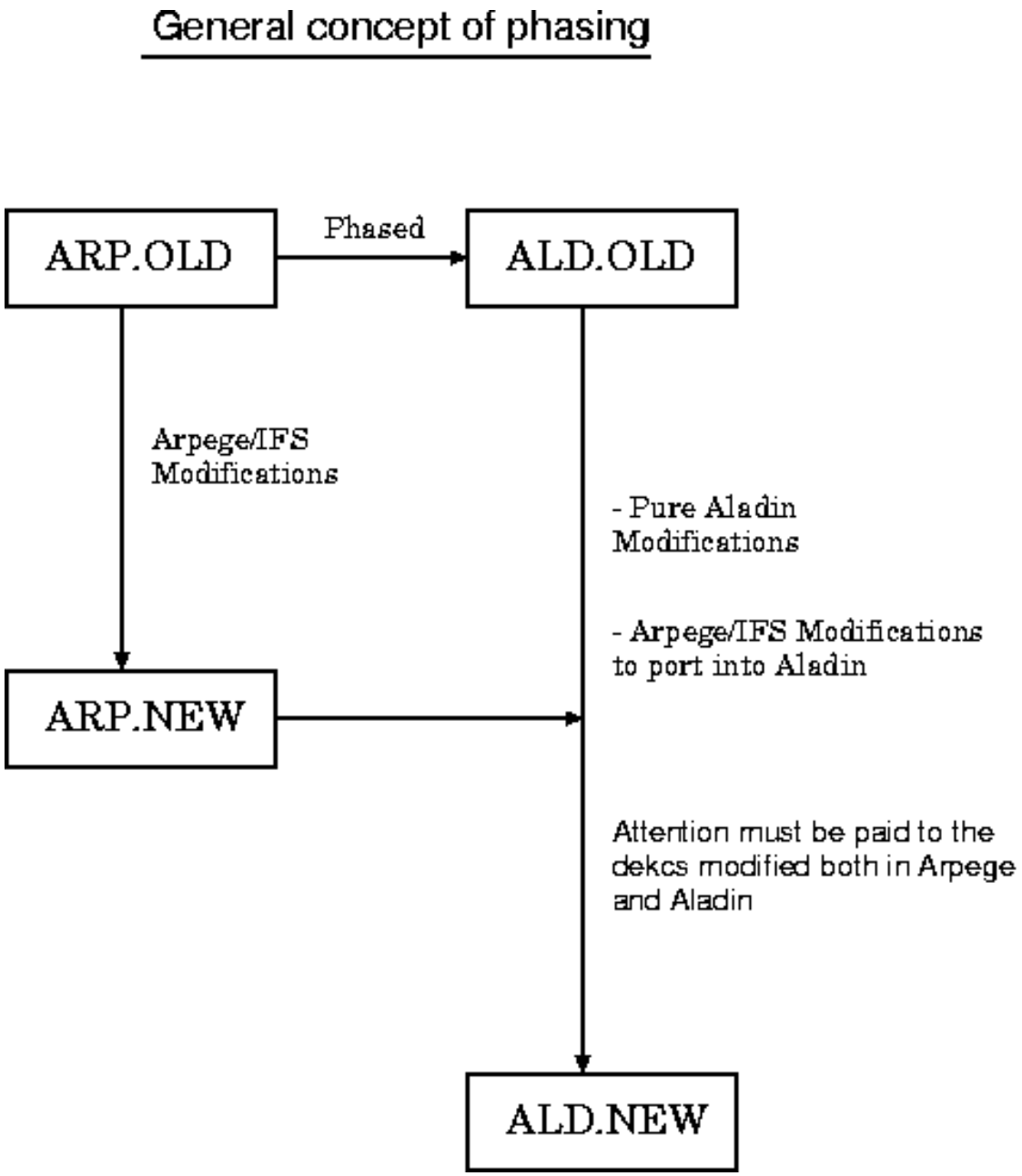
If ALADIN team decides not to port the new ARPEGE modifications, then the phaser must short-cut them by using .NOT. LELAM and/or .NOT. LRPLANE keys.

In the opposite case:

- 3 If the modified routine is pure ARPEGE one (cf Annex B.2), then the phaser must take it as it is.
- 3 If the modified routine contains a key LELAM and/or LRPLANE, the phaser must identify the modification (ClearCase tools and ARPEGE/IFS MEMORANDUM), understand it and determine its effect on ALADIN and then decide to update the old blocks under LELAM and/or LRPLANE keys.
- 3 If the modified routine is of duplicated type (cf Annex.B.2), The phaser must, first, understand the meaning of the modification, then he has to adapt it to the ALADIN context (eg. SCAN2MDM, SCAN2MDM,...). This kind of processing is too difficult to perform so a

decision must be taken by all ALADIN users.

It is good to keep in mind that comdecks and namelists common with ARPEGE/IFS, could be redefined only if ECMWF authorizes it.



IV. CODE VALIDATION.

IV.1. TYPE OF VALIDATIONS.

- After the achievement of each individual development.
- After the achievement of phasing.
- For the main configurations.
- During the double chain.

IV.2. "MAINTENANCE" VALIDATION.

- Follow-up of the step by step evolution of ARPEGE/IFS.
- Compatibility with the ARPEGE/IFS library.
- Collaboration with the << Gourous>> of SCEM/PREVI/COMPAS.
- If ARPEGE Modifications are purely technical (cleaning, internal reorganization,...), one tries to find similar results as before.

IV.3. "OPERATIONAL" VALIDATION.

- Before operational implementation.
- Emphasis on operational configurations.
- Comparison with previous operational version.
- Use of parallel suite and computation of scores on a regular basis.
- Collaboration with SCEM/PREVI/COMPAS.

IV.4. BASIC RULES OF VALIDATION.

- Respect of ARPEGE/IFS coding rules.
- Use of a validated reference.
- Inventory of modified routines, commons,... .
- Checking of namelist and all calling sequences for a given modified routine (one routine may be called at more than one place, this can be a main source of error !) .

Basic questions (after successful run):

- 3 May the tested modifications change the results?
- 3 Are there any change of default values?
- 3 Do we expect consequences on CPUtime and needed memory space?
- 3 Are there any adjoint or tangent linear counterpart to modified routines?

IV.5. VALIDATION TOOLS.

- Visualization of a selection of significant and relevant maps.
- Norm computations at different steps of the run and standard scores (individual or cumulated on several cases).
- Debugger (Totalview).
- Other tools like HRID, Wind bias software, spectra computation, VERIFPACK....

ANNEX A : LINK BETWEEN THE ARPEGE AND ALADIN CODES

- Since many parts of ARPEGE and ALADIN (for instance grid point calculations) are based on exactly the same scientific content, it was decided that ALADIN should not have a «stand-alone» library but should run from its own specific library coupled with one reference ARPEGE library. Of course both libraries may have their bug-fixes, which can be updated either separately orjointly, on a case by case basis.
- It is the responsibility of the ALADIN team to ensure the compatibility of its choices for the ALADIN library with the evolution of the ARPEGE code as reflected in the reference ARPEGE

library chosen before an ALADIN phasing action starts.
- On the other hand, anticipation of the future ALADIN evolution may be introduced in an ARPEGE (or IFS/ARPEGE) phasing action; this step, aimed at easing the future compatibility, is generally limited to «INCLUDE-type» decks.
- When it is impossible to create or to maintain the above-defined compatibility (usually because of the reduced number of working configurations in ALADIN with respect to IFS/ARPEGE) it is again the responsibility of the ALADIN team to report the necessary «fixes» from one Cycle to the next and to adapt them if needed.

ANNEX B : ROUTINES/DECKS USED IN RUNNING ALADIN.

B.1. THE DIFFERENT TYPES.

- 1) Pure ARPEGE routines/decks (sometimes coming from an ALADIN background, e.g. LAM geometry for Full-Pos). Typical example: parametrization computations.
- 2) ARPEGE routines containing «LELAM» or «LRPLANE». Typical examples: set-up routines and some semi-Lagrangian basic computation routines.
- 3) Duplicated ARPEGE routines (in a number maintained as small as feasible). Typical example: control routines CNT3 => SCAN2M DM, SCAN2MSM
- 4) ALADIN routines duplicating an ARPEGE functionality under new conditions (the duplication may be total or partial depending on the targeted use). Typical example: spectral computations (SUESCAL, ECAIN, ELDIRH, ELINVH, ...).
- 5) Specific ALADIN routines/decks. Typical example: coupling routines.

B.2. THE SPECIFICITY OF MAINTENANCE FOR EACH ABOVE-DEFINED CATEGORY.

- 1) No choice => no problem! If there is a problem this means a change of category! This does not mean that ALADIN people cannot use their own choices for the corresponding scientific problems, the changes must simply first appear in an ARPEGE library.
- 2) One has to do the following three things:
- to verify the compatibility of the changes in ARPEGE with the status of the routine; if not => change of category;

- when appropriate, to import, inside the conditional LELAM or LRPLANE blocks, the scientific equivalent of the ARPEGE modifications;
- when needed (because of too complex ARPEGE changes or because of ALADIN new specificities that cannot be treated otherwise), to create new conditional blocks.

- 3) They represent the worst phasing problem, since their very existence already shows the impossibility to work in a more «natural» fashion. No specific rules can be given here, except to leave the matter in the hand of «specialists»!
- 4) Simple in principle, but not in practice. It is not unusual that the scientific-technical basis of an ARPEGE change is either misinterpreted or insufficiently analyzed and that the ALADIN transcription works but not as it should. Conversely, if there is an ALADIN innovation that could be brought back to the benefit of ARPEGE in its next phasing, nothing guarantees that it will indeed be done and this is often becoming a source of problems in subsequent phasing actions.
- 5) No problem, except of course those of internal compatibility between the ALADIN updates, this being also true for categories 2 and 4.

ANNEX C: SPECIFIC PROBLEMS OF ALADIN «POST-PHASING» OR «BACK-PHASING»

- The problems of ALADIN post-phasing are in principle all described in the previous item, given the chosen strategy of splitting the libraries in complementary parts. The only real difficulty is the choice of the reference ARPEGE Cycle for a given ALADIN one. Thus, AL07 was phased with a specific branch on CY16T1, CY16T1_al07, while AL08 is directly phased with CY18T1 and AL09 is hoped to be in phase with CY19T1. At the beginning of the project, when the scope of the exercise was limited, they were the reference IFS/ARPEGE Cycles. Then, with complexity, the need to anticipate a few things at the level of ARPEGE meant going to interim Cycles. The flexibility offered by this new freedom was so attractive that the «natural» evolution lead to too high orders of those, until disaster stroke with 15t4 that had to be abandoned with a return to 15t2. The policy should now be to try and stick to NNt1 Cycles as much as possible.
- The special case of ALADIN Cycle 8: Cy18 will be the first cleaning cycle of ARPEGE with which ALADIN will have to be phased. It was decided to try the following strategy:

ARPEGE	ALADIN
CY13R5	AL04
CY14T3	AL05
CY15T2	AL06 (Bad !)
CY16T1+cleaning	AL07
CY18T1	AL08
CY19T1	AL09 (Hoped !)

so that CY17 (the «uncleaned» IFS/ARPEGE counterpart of CY18) will have no corresponding ALADIN Cycle!

- Back-phasing problems arise when there are successful ARPEGE developments on Cycles (full ones or interim ones) further than the one with which ALADIN is currently phased or being phased. If one either urgently needs to benefit from the corresponding enhancements or when compatibility during the coupling process imposes it, back-phasing becomes a necessity. It can be done at any level, i.e. in general either to the Cycle in creation or to the one in operational use. It requires a good knowledge of all implications of such an acrobatic endeavour and is thus reserved again to «specialists».

ANNEX D: WORKSTATION VERSION OF ALADIN (ANTICIPATED RULES OF MAINTENANCE)

The first workstation version of ALADIN was built in an ad-hoc way from a blend of post-phasing and back-phasing actions involving Cycles 4 and 5 (with a bit of Cycle 6 being subsequently added). This situation was not satisfying from a long term point of view and it was decided to restart from scratch at the occasion of Cy7. Fortunately progress in portability resulted in a rather easy phasing so that a more consistent approach can be used from now on:

- - not to try to have continuity between the workstation versions;
- - to identify and keep track of the changes needed at the last occasion in order to minimize the trouble of redoing them;
- - to identify which of these actions are platform-dependent, in order to create a kind of auxiliary library with the relevant decks (in which relevant switches can be introduced).