AT|25.1.1|038

AT PROGRAMMING STANDARDS

R.P.Norris

19 December 1986

1 INTRODUCTION

In 1985 AT programming standards were discussed [illegible] and a set of recommendations proposed. These were documented by R.H.Wand in AT/25.1.1/016, which should be consulted for the rationale behind some of the edicts issued herein. Recently, a request from the correlator group prompted a re-appraisal of those standards. The result of that re-appraisal is presented here. This document adopts wholeheartedly the philosophy of AT/25.1.1/016, and makes only a small number of changes as a result of (i) experience gained since 1985, and (ii) changes in the computing configuration for the AT. Items which have been changed since 1985 are prefaced by a (*).

It should be noted that this document says little about programming style, since we all know what the elements of good style are - don't we? Instead, this document primarily addresses the programming conventions adopted for the AT.

Attention is drawn to the ELF (ELegant Fortran) prettifier. All code should be passed through this wherever possible. See Phil Mulhall for more details on ELF. Attention is also drawn to CMS (Code Management System) installed on all RP VAXes. Use this wherever possible. To assist in this, there should normally be only one program or subprogram in each VMS file.

2 SCOPE

The programming standards presented here should be used wherever possible on all software written for the AT (*) with the following exceptions:

1. Non-VAX machines (e.g. PDP-11/73, Cyber 205)
2. software to be used in the AIPS environment, which should conform to the AIPS programming standards rather than to the AT programming standards.
3. VAX software containing Rdb instructions for the RFO pre-processor.

In these exceptional cases software should adhere as far as possible to the spirit of these standards.

3 LANGUAGE

VAX FORTRAN-77 (VF) shall be the language used. Note that

1. All Vax extensions to FORTRAN-77 are permitted, except where stated below, and

2. RATFOR is specifically prohibited (!)

Specifically, use of the following Vax extensions is ENCOURAGED, as an aid to better code:

1. All modules should contain IMPLICIT NONE.

2. INCLUDE files should be used freely, subject to the rules for include files outlined in Section 6 below.

3. PARAMETER should be used rather than coded numerical constants. (*) Each PARAMETER should be prefaced by a line of comment explaining what the parameter is.

4. Use the DO WHILE and END DO constructs, and omit the label from do-loops wherever possible.

Use of the following constructs is DISCOURAGED:

1. Computed GOTO

2. Arithmetic IF

3. ASSIGN

4. PAUSE (*) except during the development of software

5. BLOCK DATA

6. EQUIVALENCE

7. DIMENSION (include the array sizes in type declarations instead)

8. Initialisation of data in type declarations: use explicit DATA statements instead

Note also the following conventions:

1. Lower case characters if used in code should be used consistently.

2. Use explicit rather than implicit type conversions

3. Handle characters using only the VF character string mechanisms.

4. Avoid labels where possible, by using label-less DO loops and (*) embedding formats within i/o statements. Where labels are necessary, use them only on FORMAT and CONTINUE statements. They should be less than 10000 and increase monotonically through the routine.

5. Variable names should be as far as possible self-explanatory, and need not be limited to six characters.

6. Use only meaningful names

7. Define sizes of tables, arrays, and buffers parametrically.

8. Do not access uninitialised variables

9. Do not test REAL variables for equality (*) except in the special case of testing whether a REAL that has been set to zero (and thus in VAX FORTRAN has an exponent field of zero) has been reset to a non-zero quantity.

10. Wherever possible use program libraries rather than write code to duplicate the functions of existing routines. (*) Where 'in-house' library code duplicates VAX library or system service routines, use the VAX code in preference.

4 STRUCTURE

Follow the principles of structured programming, including the concept of top-down design. Write out the specification first, then the modules' prefatory commments and calling sequence, and finally write the code. Use the following guidelines:

1. Modularise rather than write large monolithic single programs.

2. Avoid the use of GOTO

3. Avoid 'internal subroutines' driven by GOTO statements.

4. Put COMMONs and data structures, (*) along with their type declarations and comments, in INCLUDE files.

5. Eradicate unreachable code, unused labels, and unused declarations.

6. Code for clarity rather than efficiency

5 PRESENTATION

1. All program and subprogram units must begin with a PROGRAM, FUNCTION, or SUBROUTINE statement, and finish with an END statement.

2. (*) Each subprogram should be stored in a VMS file with the same name as the subprogram (but possibly prefixed with a common prefix for one package). Only one subprogram should normally be stored in each file. On any occasion where more than one subprogram is in a file, each subprogram should be clearly distinguished from the preceding subprogram by, for example, a number of blank lines followed by a line of asterisks.

3. Each subprogram should start with a block of prefatory comments. These should include the author, date, functional description, list of arguments (along with their function and type), implicit inputs (from files or COMMONs), implicit outputs, and subroutines called. Any modifications made to the program should also be listed here with a date and author, (*) even if the code is being cared for by CMS.

4. Comments should be clearly distinguishable from code and accurately reflect the behaviour of the program. They should be generous without stating the bleedin' obvious and must precede the code they describe.

5. Program structure should be indicated by a consistent indentation structure. (*) This is best achieved by passing all code through ELF. To aid this process, comments must have a C in column 1 rather than a *, continuation lines should have a : in column 5, and labelled DO-loops should be avoided.

6 INPUT AND OUTPUT

1. When convenient, FORMATs should be embedded in the read or write statements. Failing that, FORMAT statements should be placed immediately after the corresponding i/o statement (if used only once) or else at the end of the module.

2. External unit identifiers should be defined using LIB$GETLUN.

3. Terminate input by end-of-file rather than count.

4. (*) Test for an error condition on input.

5. Applications should not distinguish between upper and lower case in data they recieve. Use STR$UPCASE where necessary to achive this.

6. Validate everything that comes into a program from outside.

7 SUBPROGRAMS

1. (*) FUNCTION subprograms should have the type declaration in the FUNCTION statement rather than in the body of the code.

2. (*) Arguments of subprograms should have the type declaration immediately after the FUNCTION statement, before all other type declarations.

3. Arguments should be in the order: given, given and altered, returned, status. (*) Wherever possible, 'given and altered' parameters should be avoided altogether.

4. Do not mix data types across a subprogram call

8 INCLUDE FILES

1. (*) Each INCLUDE file must start with a brief description of the contents.

2. (*) Each item in the include file should be accompanied by a line of explanation.

3. (*) Each include file should be self-contained wherever possible in the sense that it contains the type declarations, commons (where applicable), and array dimensions (e.g. Parameter) of each item.

9 ERROR HANDLING

Leave error handling to the calling program wherever possible by returning a status. This status should either be:

1. An INTEGER conforming to the VAX convention for error conditions

2. (*) A CHARACTER string containing 'OK' for no error, or else a description of the error.

10 (*) QUANTITIES (ABSTRACTED FROM AT/16.3.1/032)

10.1 Positions

The AT will operate internally in J2000 coordinates only. However, observers may if they wish specify their source coordinates in B1950 or other frames, but these will immediately be precessed on input to J2000. All AT software will store and use RA and Dec in J2000 radians, stored internally as REAL*8.

10.2 Units

All quantities within AT software must be specified in SI units, and all angles must be specified in radians. Thus arcsec, mm, cm, minutes, deg/sec, etc., are all illegal. However, the quantities may be converted to 'friendly' units at the interface to the outside world, but this conversion must take place right at the i/o stage. Thus the observer may specify a 1 degree source offset (or the engineer a 1mm telescope location offset) into his terminal, but these must immediately be converted to SI units (radians and metres respectively) before being passed to other software.

10.3 Floating Point Formats

At present, the VAX 750's execute D-floating far more efficiently than G-floating. However, newer VAXes will execute G-floating by default. Thus at present routines should be compiled using D-floating, but this convention will have to be changed at some point in the near future.