TECHNICAL MEMORANDUM

52

The DOCTOR system. A DOCumenTary ORiented programming system

J.K. Gibson

Research Department

January 1982

1. INTRODUCTION

DOCTOR is a programming standard born out of the advantages and shortcomings found by the author during his experience with other programming standards. Two standards in particular have been used extensively - the OLYMPUS system (Roberts, 1974) and the Control Data Corporation programming standard. In addition, many useful ideas and concepts are based on Frank's (1971) documentation processor. (DOCK, an Internal/External Documentation Processorm is a CDC Proprietary Product, copyright Control Data Corporation, T971). Since the DOCTOR system was first described (Gibson, 1980), it has been developed and modified, and is currently being used for ECMWF's new forecasting system. DOCTOR's application will be described with particular reference to FORTRAN. There is no reason why, suitably modified, some or all of its principles cannot be used in conjunction with any programming language.

1.1 Basic aims of the system

DOCTOR attempts to:

a) provide well presented code.

b) produce source code following a standard structure.

c) set up points of reference for external documentation.

d) enable the inclusion within the source code of documentation which can be extracted mechanically.

e) allow maximum communication between routines by storing universal variables in structured pools or common blocks.

f) facilitate the recognition of variable types, and the differentiation between local variables, variables in common blocks, and dummy arguments to routines.

g) provide a set of utility routines for copying vectors, resetting arrays, etc.

1.2 Documentation

Documentation is a means of retaining the ability of code to be understood. The production of documentation is a skill at least as important as the skill required to design and generate the code itself. Good documentation increases the value of code - it assists maintenance, aids understanding, and can be invaluable if language to language re-coding should ever be necessary. One of the biggest problems with documentation is keeping it up to date. When documentation takes the form of typed or printed material there is always a delay in producing amendments. There is also a danger of changing the code, but not the documentation. This danger may be reduced if much of the documentation is incorporated in the text of the source code. If facilities are provided to extract the documentation from the source code, delays in producing amendments are overcome, and labour costs reduced. The DOCTOR system provides a simple set of conventions to enable sufficient documentation to be included in the souce code of each routine to enable it to be understood and used by another user. These conventions are such that simple extraction programmes may be written to process the source code and produce the documentation. In addition, routines are structured into numbered sections and sub-sections, thus facilitating cross references to lists of equations and symbols etc., which could not easily be included in the source code. Such additional documentation should be detailed in the section of the header comments titled REFERENCE (See 2.3.1 below).

2. CODING CONVENTIONS

Code should follow a modular structure, each module or routine fulfilling a stated purpose. Modules should be divided into numbered sections and sub-sections. Communication between modules should not involve long parameter lists - shared data should be made available through shared data pools, common blocks, etc.

2.1 Naming conventions

The type of a variable is indicated by the first letter of the variable's name according to Fig. 1.

PREFIX	**TYPE**
I,J,K,M,N	INTEGER
H	INTEGER (used for HOLLERITH)
L	LOGICAL
Y	CHARACTER
ALL OTHERS	REAL

Fig. 1 Variable types

In addition, a prefix convention is used to indicate the STATUS of a variable, to enable recognition of local variables, COMMON variables, loop control variables, dummy arguments, parameters, etc., according to Fig. 2.

A "Hollerith" type is included for ease of communicating character data via COMMON blocks and including characters in records to be read or written via unformatted input/output. Conversion between "Hollerith" and CHARACTER type would be performed in FORTRAN by means of internal files.

The default DOCTOR type convention may be established in FORTRAN by the statement:

```
IMPLICIT INTEGER (H), LOGICAL (L), CHARACTER *8 (Y)
```

STATUS	**TYPE**	**PREFIX**
LOCAL VARIABLES	INTEGER	I
	REAL	Z
	INTEGER (HOLLERITH)	HO
	LOGICAL	LO
	CHARACTER	YO
DUMMY ARGUMENTS	INTEGER	K
	REAL	P(But not PP)
	INTEGER (HOLLERITH)	HA
	LOGICAL	LA
	CHARACTER	YA
LOOP CONTROL	INTEGER	J(But not JP)
PARAMETER	INTEGER	JP
	REAL	PP
	LOGICAL	LP
	CHARACTER	YP
GLOBAL OR COMMON	INTEGER	ANY INTEGER PREFIX NOT INCLUDED ABOVE
	REAL	ANY REAL PREFIX NOT INCLUDED ABOVE
	INTEGER (HOLLERITH)	H, BUT NO HO, HA,OR HP
	LOGICAL	L, BUT NOT LO, LA, OR LP
	CHARACTER	Y, BUT NOT YO, YA, OR YP

Fig. 2 Variables by status

The naming conventions enable the type and status of each variable in the code to be recognized from its name.

2.2 Conventions for comments

All comments within the code should take the form of titles, clauses, or sentences terminated by full stops or periods. Abbreviations terminated by full stops (e.g. etc.) should be either avoided or coded without the full stop (eg etc). Proper names appearing in the middle of a clause or sentence should be prefixed by an asterisk (*), indicating an upper case first letter to the document extraction programme. Alternatively, proper names may be parenthesised by asterisks (eg *FORTRAN*) to indicate that the whole word is to be extracted as upper case letters. Header titles and section titles should be "underlined" by coding - under each character in the next consecutive statement (including punctuation characters). Underlined titles will be recognised by the document extraction programme and printed in upper case. A single letter followed by a full stop will be assumed to be an initial, and extracted as upper case.

With this convention, the following rules apply to an upper/lower case documentation extraction programme.

a) The first letter after each full stop is UPPER CASE.

b) Words that are underlined are UPPER CASE.

c) Words backeted by asterisks are UPPER CASE.

d) Words prefixed by a single asterisk have first letter UPPER CASE only.

e) Single letters followed by full stops are UPPER CASE.

f) All other letters are LOWER CASE.

2.3 Code structure

Routines are divided into a preamble or "header", sections, and subsections. The header contains comments which describe the routine; the main body of the code is divided into numbered sections and subsections.

2.3.1 Header comments

The comments at the head of each routine should take the form of

a) A C**** card containing a title beginning with the subroutine name.

b) The name of the author and the date written

c) Modification details (name, date, and reason for modification)

d) headed sections, giving the following information:

```
PURPOSE.        the function of the routine
--------

INTERFACE.      how the routine receives its data and returns its
----------      results.

METHOD.         how the results are obtained
-------

REFERENCES.     references to external documentation, if any.
-----------
```

Each headed section begins with a comment card containing the title key-word followed by a full stop. The following card should underline the title and the full stop.

The headed section following the PURPOSE section should contain C** in columns 1 to 3. This enables the header comments up to this section to be extracted as OVERVIEW documentation, while the complete header comments would be extracted as EXTERNAL or INTERNAL documentation. (OVERVIEW, EXTERNAL and INTERNAL levels of documentation are defined in Section 3). An example of a routine header is given in Fig. 3.

```
      SUBROUTINE EG1(HATEXT,KLEN)
C
C**** *EG1* - ROUTINE TO PRINT A TEXT ARRAY.
C
C     J.K.GIBSON      E.C.M.W.F.      13/1/82.
C
C     MODIFIED BY     J.K.GIBSON      16/1/82  - TO DEMONSTRATE THE WAY
C                                                IN WHICH CHANGES SHOULD
C                                                BE DOCUMENTED.
C
C     PURPOSE.
C     --------
C
C          *EG1* PRINTS A *HOLLERITH* ARRAY CONTAINING 8 CHARACTERS PER
C     WORD.
C
C**   INTERFACE.
C     ----------
C
C          *CALL* *EG1(HATEXT,KLEN)*
C
C               *HATEXT*    - *HOLLERITH* ARRAY CONTAINING TEXT.
C               *KLEN*      - LENGTH OF *HATEXT.*
C
C     METHOD.
C     -------
C
C          *HATEXT* IS PRINTED USING FORMAT *(10A8).*
C
C     EXTERNALS.
C     ----------
C
C          NONE.
C
C     REFERENCE.
C     ----------
C
C          NONE.
C
```

Fig. 3. Routine header

The header comments should normally be followed by

a) an IMPLICIT statement

b) common blocks

c) the main body of the code.

2.3.2 Common blocks

Common blocks should begin with a C* card containing a title beginning with the common block name. This should be followed by the name of the author and the date written, and modification details. If an editor capable of allowing conditional code is available, a section of comments can be conditionally included giving a table of information under the headings:

```
C*      VARIABLE        TYPE        PURPOSE
C       --------        ----        -------
```

It may also be considered desirable to include the FORTRAN source code of some common blocks in the extracted external or internal level documentation. This is indicated by bracketing such code by C*** cards. Common blocks should be "ruled off" from one another by including a comment card containing minus signs in columns 7 to 72 at the end of each block. Figure 4 illustrates the format for a common block. The CDC UPDATE conditional statements *IF DEF, DOC and *ENDIF may be used to restrict the inclusion of the table of variables, and the inclusion of the C*** brackets unless "DOC" is defined to UPDATE. A common block with conditional statements is illustrated in Fig. 5.

```
C
C*     *COMMON* *COMEG1* - EXAMPLE OF A COMMON BLOCK.
C
C      J.K.GIBSON      E.C.M.W.F.      13/1/82.
C
       COMMON /COMEG1/
     I NDVIN,     NDVOUT,     HMSG
C
C*     VARIABLE       TYPE        PURPOSE.
C      --------       ----        --------
C
C      *NDVIN*       INTEGER     LOGICAL UNIT FOR INPUT.
C      *NDVOUT*      INTEGER     LOGICAL UNIT FOR OUTPUT.
C      *HMSG*        HOLLERITH   ARRAY CONTAINING A TEXT MESSAGE.
C
       DIMENSION
      I HMSG(10)
C
C      ----------------------------------------------------------------
```

Fig. 4. Common block format

```
C
C*     *COMMON* *COMEG1* - EXAMPLE OF A COMMON BLOCK.
C
C      J.K.GIBSON      E.C.M.W.F.      13/1/82.
C
C      MODIFIED BY     J.K.GIBSON      16/1/82   -  TO ILLUSTRATE HOW *CDC*
C                                                   *UPDATE* DIRECTIVES MAY
C                                                   BE USED TO CONTROL THE
C                                                   DOCUMENTATION PRODUCED.
C
*IF DEF,DOC
C***
*ENDIF
       COMMON /COMEG1/
     I NDVIN,     NDVOUT,     HMSG
*IF DEF,DOC
C***
C*     VARIABLE       TYPE        PURPOSE.
C      --------       ----        --------
C
C      *NDVIN*       INTEGER     LOGICAL UNIT FOR INPUT.
C      *NDVOUT*      INTEGER     LOGICAL UNIT FOR OUTPUT.
C      *HMSG*        HOLLERITH   ARRAY CONTAINING A TEXT MESSAGE.
C
C***
*ENDIF
       DIMENSION
      I HMSG(10)
*IF DEF,DOC
C***
*ENDIF
C
C      ----------------------------------------------------------------
```

Fig. 5. Common block with conditional statements (CDC update form)

2.3.3 Main body of the code

The main body of the code should be divided into sections. Each section should be numbered, and should begin with

a) A C* card containing the section number and title

b) A comment card underlining the title.

c) A CONTINUE statement numbered in accordance with the section number. (i.e. N00 for section N).

eg

```
C
C*          1.      SET INITIAL VALUES.
C                   --- ------- -------
C
  100 CONTINUE
```

Sections should be "ruled off" from one another by a comment card containing minus signs in columns 7 to 72.

Sections should be divided into subsections. Subsections should be numbered such that N.M is subsection M of section N. Each subsection should begin with:

a) A C* card containing the subsection number and title

b) A CONTINUE statement numbered NM0.

eg

```
C
C*          1.1     SET LOGICAL SWITCHES.
  110 CONTINUE
```

2.3.4 Additional comments

Blank comment cards (ie C in column 1 only) should be included as necessary to improve the layout of documentation and to enhance the readability of the code. Very few additional comments will be necessary, as a listing of the section and subsection titles should represent the steps in the coded algorithm. Additional comments are useful to document branches in the code, and are necessary in the case of any unusual coding construction. The aim should be to provide comments which, when extracted, list the algorithm coded step by step. Cross references should be to section and subsection numbers, not to statement numbers in the code

eg

```
          GO TO 300
C*                        BRANCH TO 3.
```

Figure 6 illustrates the code structure defined.

```
      SUBROUTINE EG2(KDVIN,KDVOUT)
C
C**** *EG2* - ROUTINE TO CONTROL A PROCESS.
C
C     J.K.GIBSON      E.C.M.W.F.     13/1/82.
C
C     MODIFIED BY     J.K.GIBSON     16/1/82  -  CALLS TO *SUB2* WERE
C                                                ADDED TO EXTEND THE
C                                                SCOPE OF THE ROUTINE.
C
C     PURPOSE.
C     --------
C
C          *EG2* CONTROLS A PROCESS. IT IS AN EXAMPLE OF THE *DOCTOR*
C     PROGRAMMING SYSTEM.
C
C**   INTERFACE.
C     ----------
C
C          *CALL* *EG1(KDVIN,KDVOUT)*
C
C               *KDVIN*     - LOGICAL UNIT FOR INPUT DATA.
C               *KDVOYUT*   - LOGICAL UNIT FOR OUTPUT DATA.
C
C     METHOD.
C     -------
C
C          VARIOUS SUBROUTINES ARE CALLED TO PERFORM THE TASKS REQUIRED.
C
C     EXTERNALS.
C     ----------
C
C          *SETCOM* - SUBROUTINE TO SET DEFAULT VALUES.
C          *GETP*   - SUBROUTINE TO DECODE INPUT PARAMETERS.
C          *SUB1*   - SUBROUTINE TO PERFORM PART 1 OF PROCESS.
C          *SUB2*   - SUBROUTINE TO PERFORM PART 2 OF PROCESS.
C          *OUTPUT* - SUBROUTINE TO PRINT RESULTS.
C
C     REFERENCE.
C     ----------
C
C          A DESCRIPTION OF THE PROCESS MAY BE FOUND ON FILE 234,
C     NUMBER 567.
C
      IMPLICIT INTEGER(H),LOGICAL(L),CHARACTER*8(Y)
C
C*    *COMMON* *COMEG2* - VARIABLES USED IN THE COMPUTATION.
C
C     J.K.GIBSON      E.C.M.W.F.     13/1/82.
C
      COMMON /COMEG2/
     I NUM1,     NUM2,     NUM3,
     R COEF1,    COEF2,    COEF3,
     L LFLAGS
C
C*    VARIABLE      TYPE      PURPOSE.
C     --------      ----      --------
C
C      *NUM1*      INTEGER    NUMBER OF TYPE 1 PRODUCTS.
C      *NUM2*      INTEGER    NUMBER OF TYPE 2 PRODUCTS.
C      *NUM3*      INTEGER    NUMBER OF TYPE 3 PRODUCTS.
C      *COEF1*     REAL       COEFFICIENT FOR TYPE 1 PRODUCT.
```

```
C       *COEF2*    REAL        COEFFICIENT FOR TYPE 2 PRODUCT.
C       *COEF3*    REAL        COEFFICIENT FOR TYPE 3 PRODUCT.
C       *LFLAGS*   LOGICAL     LOGICAL SWITCHES.
C
      DIMENSION
     L LFALGS(20)
C
C      ----------------------------------------------------------------
C
C*          1.      OPEN FILES.
                    ---- ------
C
C
  100 CONTINUE
      OPEN(NDVIN)
      OPEN(NDVOUT,STATUS='OLD')
C
C      ----------------------------------------------------------------
C
C*          2.      CONTROL THE PROCESS.
                    ------- --- --------
C
C
  200 CONTINUE
C
C*          2.1      SET DEFAULT VALUES AND DECODE PARAMETERS.
  210 CONTINUE
      CALL SETCOM(NDVIN)
      CALL GETP
C
C*          2.2      PART 1 OF THE PROCESS.
  220 CONTINUE
      CALL SUB1
      IF (.NOT.LFLAGS(1)) GO TO 220
C*                   BRANCH BACK TO 2.2 FOR NEXT ITERATION IF PART 1
C*                   HAS NOT CONVERGED.
  222 CONINUE
C
C*          2.3      PART 2 OF THE PROCESS.
  230 CONTINUE
      CALL SUB2
      IF (.NOT.LFLAGS(2)) GO TO 230
C*                   BRANCH BACK TO 2.3 FOR NEXT ITERATION IF PART 2
C                    HAS NOT CONVERGED.
C
  232 CONTINUE
C
C      ----------------------------------------------------------------
C
C*          3.      PRINT THE RESULTS.
                    ----- --- --------
C
C
  300 CONTINUE
      CALL OUTPUT(NDVOUT)
C
C      ----------------------------------------------------------------
C
C*          4.      CLOSE FILES.
                    ----- ------
C
C
  400 CONTINUE
      CLOSE(NDVIN)
      CLOSE(NDVOUT)
C
C      ----------------------------------------------------------------
C
      RETURN
      END
```

Fig. 6. Subroutine coded to DOCTOR standard

3. EXTRACTION OF DOCUMENTATION

Some of the coding conventions defined in Section 2 are intended to enable documentation to be extracted by means of a fixed set of rules. Using these rules it is possible to write a simple documentation extraction programme to process the source code and extract the documentation. A programme called DOC is available at ECMWF for this purpose.

3.1 Levels of documentation

Four levels of documentation are defined - OVERVIEW, SPECIAL, EXTERNAL, and INTERNAL. Users may request three levels - OVERVIEW, EXTERNAL or INTERNAL. Sections of documentation within the source code at each lvel begin with a "trigger" and end with a "terminator".

3.1.1 Overview documentation

Overview documentation is triggered by a C**** card, and terminated at the first subsequent non-comment card, unless a trigger for another documentation level is encountered first. Normally overview documentation will contain the title and purpose of each routine.

3.1.2 Special documentation

Special documentation is triggered by a C*** card, and terminated by a subsequent C*** card. The special level of documentation is used to enable source code other than comments to be extracted as part of EXTERNAL or INTERNAL documentation. It should be used to allow the extraction of code which sets the values of critical constants, etc.

3.1.3 External documentation

External documentation is triggered by a C** card, and terminated at the next subsequent triggger or non-comment card. It is assumed that a request to extract external documentation implies the extraction of overview and special documentation as well. The documentation extracted at the external level is intended to provide sufficient information for an external user to use the routines documented.

3.1.4 Internal documentation

Internal documentation is triggered by a C* card, and terminated at the next subsequent trigger or non-comment card. A request to extract internal documentation implies the extraction of overview, special, and external documentation as well. The internal documentation should contain the basic steps of the algorithms coded, and should be sufficient to enable maintenance and modification of the code.

4. UTILITY ROUTINES

The concept of utility routines is taken from the OLYMPUS system (Roberts, 1974). The purpose of utility routines is to provide the user with a set of tools for performing common routine tasks, such as copying or re-setting arrays.

With the advent of the X3J3 FORTRAN 77 standard, formatted input/output in FORTRAN has been considerably simplified, especially where text strings are involved. Thus many of the standard OLYMPUS utilities are not included in the basic set of utilities for DOCTOR. (It seems pointless to inclue a routine such as

```
CALL MESSAGE (48H text of message)
```

to perform the single statement

```
WRITE (NDVOUT, '(A)') 'text of message')
```

Figure 5 lists a basic set of DOCTOR utility routines, their CALL sequence, and their purpose. This basic set should be available to users of the DOCTOR system. It may be supplemented as desired.

CALL SEQUENCE	**PURPOSE**
CALL RESETR (PA,KLEN,PVAL)	Set the first KLEN value of PA to PVAL.
CALL RESETI (KA,KLEN,KVAL)	Set the first KLEN values of KA to KVAL.
CALL RESETL (LA,KLEN,LAVAL)	Set the first KLEN values of LA to LAVAL
CALL RESETH (HA,KLEN,HAVAL)	Set the first KLEN values of HA to HAVAL
CALL COPYR (PA,K1,PB,K2,KLEN)	Copy KLEN values from PA(K1) to PB(K2).
CALL COPYI (KA,K1,KB,K2,KLEN)	Copy KLEN values from KA(K1) to KB(K2).
CALL COPYL (LAA,K1,LAB,K2,KLEN)	Copy KLEN values from LAA(K1) to LAB(K2).
CALL COPYH (HAA,K1,HAB,K2,KLEN)	Copy KLEN values from HAA(K1) to HAB(K2).
CALL SCALER (PA,KLEN,PB)	Scale the first KLEN values of PA by PB.
CALL SCALEI (KA,KLEN,KB)	Scale the first KLEN values of KA by KB.
CALL SIGNR (PA,KLEN)	Negate the first KLEN values of PA.
CALL SIGNI (KA,KLEN)	Negate the first KLEN values of KA.

Fig.7 DOCTOR utility routines

REFERENCES

Frank, R.H. 1971 DOCK - an INTERNAL/EXTERNAL documentation processor (Copyright Control Data Corp.). (Extracted from the source code by courtesy of Frank Stevens, CDC).

Gibson, J.K. 1980 Programming Systems, Docmentation, OLYMPUS,, DOCTOR. ECMWF Tech.Memo.No.20, pp.61.

Roberts,K.V. 1974 The Olympus Programming System Computer Physics Communication, 7, 237-240.

Current Status of Meteorological Bulletins

The following indicates the status of Meteorological Bulletins at 29.10.1991:-

Reference	Title	Status
M0.	**General**	
M0.0	Introduction	Not issued
M1.	**Basic Functional Design of ECMWF's Meteorological Operational System (EMOS)**	
M1.0	General description	
M1.0/1	Current and Planned Meteorological Applications Systems at ECMWF	10/85 original version
M1.0/2	Standards for Software Development	06/86 oeiginal version
M1.1	Data Bases	
M1.1/1	General	Not issued
M1.1/2	Message Data Base	Not issued
M1.1/3	Reports Data Base	Not issued
M1.1/4	Fields Data Base	01/87 original version (needs updating)
M1.1/5	Products Data Base	Not issued
M1.1/6	Data Base for Trajectory Models	10/91 revision 1
M1.2	Process control	Not issued
M1.3	Data Acquisition	
M1.3/1	General	06/87 original version
M1.4	Pre-processing	
M1.4/1	General	09/89 original version
M1.4/2	Decoding	01/91 original version
M1.4/3	Data checking and validation	05/90 original version
M1.5	Analysis	See Research Manuals
M1.6	Forecast	See Research Manuals
M1.7	Post-processing	
M1.7/1	Post-processing and dissemination	06/87 original version
M1.8	Dissemination	moved to M3.
M1.9	Archives	
M1.9/1	General	Not issued
M1.9/2	MARS User Guide	07/91 revision 9
M1.10	Metview	Not issued

M1.11	Data Monitoring	Not issued
M2.	**Guide to the ECMWF Forecasting System**	**See Research Manuals**
M3.	**The dissemination of ECMWF products**	
M3.0	Introduction to ECMWF's dissemination service	07/91 original version
M3.1	The dissemination of ECMWF products to Member States	07/91 original version
M3.2	User Guide to ECMWF products	in preparation; until ready, see ECMWF User Guide to ECMWF products version 1.1