How to implement MODULEF

Patrick Laug

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports Techniques

N° 69

HOW TO IMPLEMENT MODULEF

Patrick LAUG

Avril 1986

HOW TO IMPLEMENT MODULEF

Patrick LAUG

March 1986

RESUME

Cette brochure décrit la mise en oeuvre de Modulef, qui est une bibliothèque "éléments finis" contenant plus de 2 000 sous-programmes Fortran.

ABSTRACT

This notice explains how to implement Modulef, which is a finite element library including more than 2 000 Fortran subroutines.

FOREWORD

This notice is translated from Modulef Report n°83, "Mise en oeuvre de Modulef, version 85", May 1985, by Patrick LAUG. Although this latter report is updated every year, most of the specifications described here still remain valid.

SUMMARY

Appendices

1. INTRODUCTION

The purpose of this notice is to explain practically how to implement a magnetic tape which has been provided by the Modulef Club, and to describe the logical contents of such a tape.

Details concerning specific systems (Bull/Multics, CDC, IBM) are stated in appendices.

To insure portability, the provided tapes may be of two main types :

- **Multicomputer tape** : a tape of that kind is written by the INRIA computer (Bull/Multics), but it may be read on any site*. Such a tape includes only one file of characters, representing successively subroutines, data, documentation, ... Then, the addressee must split it into several libraries, compile, link, ... A detailed description of such a tape and its implementation are explained in the following paragraphs, and additional information may be given in the appendices.
- **Specific tape for a given computer** : a tape of that kind is written and read by similar computers (e.g. transfer from Multics to Multics). It may contain partitioned files, source, object, linked libraries, as also unformatted files for sequential or direct access. Implementing such tapes is explained in the appendices.

---

* **Minimal configuration** : the core needed to execute Modulef is proportional to the number of nodes in the mesh. Problems with about 2 000 nodes have been handled on Apollo DN 600, whose main features are :

32 bits VLSI CPU
1 Mbyte main memory
16 Mbyte virtual memory
20 Mbyte hard disk

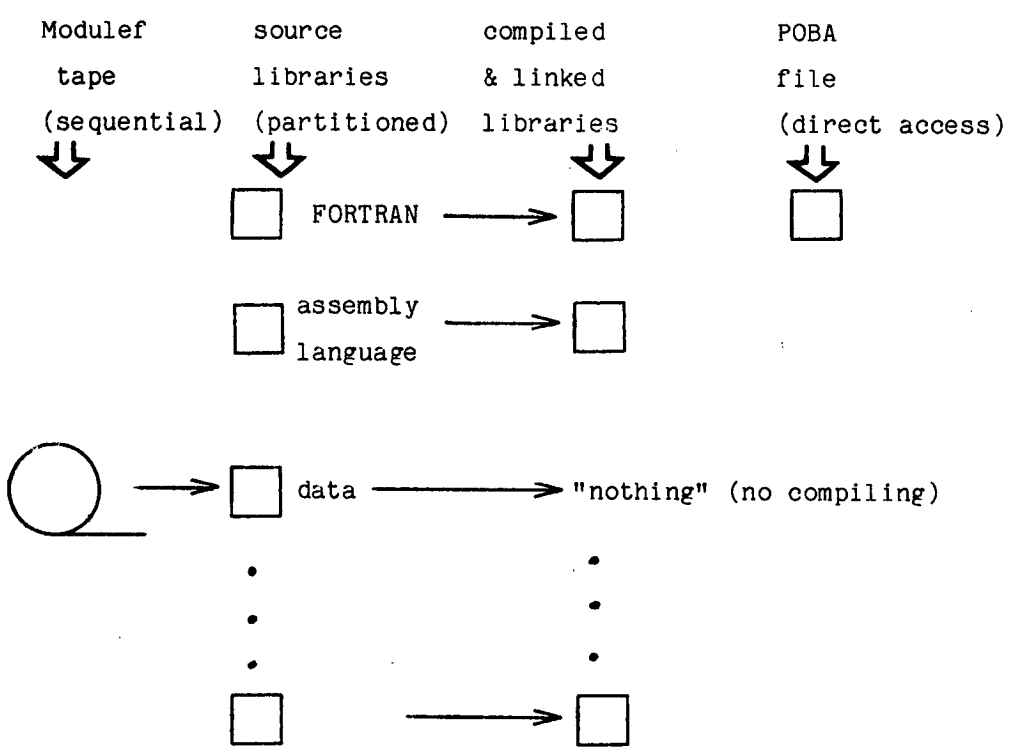
Modulef requires a Fortran 77 compiler having the two following extensions :

- if an array of type INTEGER is the actual argument of a subroutine, the corresponding dummy argument may be an array of type REAL or DOUBLE PRECISION,
- a variable of type INTEGER must be large enough to contain 4 characters (generally, 32 bits or more).

Modulef runs correctly on various computers such as Apollo, Bull/Multics, CDC, Cray, Data General, Harris, IBM, ICL, MODCOMP, Univac, Vax, ...

2. GENERAL DESCRIPTION OF THE IMPLEMENTATION OF A MULTICOMPUTER TAPE

The following figure shows how to implement a multicomputer tape :



Here is a general description of the main steps to take :

1. Read the tape. We assume here that the result is to create an intermediate disk file named ALL.
2. Split the ALL file into subfiles (one subfile per library).
3. Put each library in the form of a partitioned file (one member per subroutine).
4. Write, if necessary, the nonportable utilities.
5. Compile each library.
6. Link each library.
7. Create the procedures data base.
8. Execute program POBAXX, which generates the direct access file POBA.
9. Run the tests.
10. Use normally the Modulef libraries, expecting future updatings.

All these steps are detailed in this notice.

To avoid a tedious work, it is convenient to use automatic tools. As an example, the portable program DECIEB is introduced in this notice. More specific tools have already been written for the following computers :

BURROUGHS	by	UNAM (Mexico)
CDC		EPFL (Lausanne, Switzerland)
Bull/Multics		INRIA (Rocquencourt, France)
IBM		SNCF (Paris, France)
UNIVAC		CCSA (Bruz, France)
VAX		LSRH (Neuchâtel, Switzerland)

Such tools make it possible to group several steps, for instance 1, 2 and 3 (generating partitioned files by directly reading the tape, without creating intermediate files).

To read a Modulef tape, its physical features and logical structure must be known. Part of this information is given on the "tape writing listing" attached to each tape. Here is an example :

```
sgf -ac -sort

Nom absolu de la liste : >udd>Modulef>Laug>abs>demo

Description du fichier de sortie :

  ************************
  * tape_ibm_     MEF00  *
  * -block        7920   *
  * -mode         ebcdic *
  * -density      6250   *
  * -create               *
  * -no_labels            *
  * -format       fb     *
  * -number       1      *
  * -record       80     *
  * -retain       none   *
  ************************

Mounting volume MEF00 with a write ring.
MEF00 mounted on tape_01.

>udd>Modulef>v>s>bib1.indep.fortran.s.archive
./BIBLIF77 ADD NAME=MEFBIB1S,SSI=86022413
1.   8 membres, 125 lignes.        Cumul : 8 membres, 125 lignes.

>udd>Modulef>v>s>bib2.indep.fortran.s.archive
./BIBLIF77 ADD NAME=MEFBIB2S,SSI=86022413
2.   9 membres, 179 lignes.        Cumul : 17 membres, 304 lignes.

>udd>Modulef>v>s>bib3.indep.fortran.s.archive
./BIBLIF77 ADD NAME=MEFBIB3S,SSI=86022413
3.   56 membres, 7836 lignes.      Cumul : 73 membres, 8140 lignes.
r 13:36 26.042 347
```

More precise details about this listing are given in the following paragraphs.

3. READING A MULTICOMPUTER TAPE (STEP 1)

The physical features of the tape are indicated on the "tape writing listing", in the frame that follows the line "Description du fichier de sortie" ("Description of the output file"). In our example :

tape_ibm MEF00 means that the tape, named MEF00, conforms to the usual IBM standards (odd parity, 9 tracks, etc...). Therefore, it can be read on most of the computers (IBM or Bull, CDC, Univac, Vax, etc),

-block precedes the block length, in characters (7 920 here),

-mode precedes the encoding mode (ebcdic or ascii),

-density precedes the density, in bytes per inch (6 250 bpi here),

-create means that the previous contents of the tape have been overwritten,

-no_labels means that the tape is unlabeled and then it is possible to read it anywhere,

-format fb means "fixed blocked format" : each block contains several logical records having the same length,

-number 1 means that the file included in the tape is at first position,

-record 80 means that each logical record is made of 80 characters (here, a block contains consequently 7 920 / 80 = 99 logical records),

-retain none means that the tape was detached after writing.

Examples to read such a tape under various systems are given in appendix.

After the previous physical features, one can find some information about each library. For instance, the lines :

```
>udd>Modulef>v>s>bib3.indep.fortran.s.archive
./BIBLIF77 ADD NAME=MEFBIB3S,SSI=86022413
3. 56 membres, 7 836 lignes.    Cumul : 73 membres, 8 140 lignes
```

mean that the provided library is identified on Multics of INRIA by >udd>Modulef>v>s>bib3.indep.fortran.s.archive. The line ./BIBLI... (located on the tape at the top of the library) shows that it has been programmed in Fortran 77, is identified outside by MEFBIB3S (or BIB3 to abbreviate), and has been written on February 24, 1986 at 1 p.m. This library, which is the third one, includes 56 members making 7 836 lines. The number of members and lines is added to those of the preceding libraries on the tape (and printed after the word "cumul").

The number of lines is related to the size of the files that are created after reading. On systems where the disk-space must initially be reserved, think that the file may be extended in the future for possible updatings (avoid RELEASE on IBM).

The total number of logical records is the last number of lines after "cumul". From this number, a simple division gives the number of physical blocks. In our example, there are 8 140 logical records, i.e. 82 blocks with 99 records and one more block with 22 records (8 140 = 82 * 99 + 22).

4. SPLITTING INTO LIBRARIES (STEPS 2 AND 3)

On the tape is only one sequential file. However, the logical structure of this file allows you to split it automatically into several partitioned files. As a matter of fact :

Each library begins with a line of the form :

./BIBLIxxx ADD NAME=yyy,SSI=yymmddhh comment

Each member begins with a line of the form :

C/MEMBRxxx ADD NAME=yyy,SSI=yymmddhh comment

xxx gives the type of the library or the member :

F77 Fortran 77	DOC documentation
PL1 PL/1	JCL job control language
ASM assembly language	DIV miscellaneous
DAT data	

yyy name of the library or the member (at most 8 characters). In the case of a library, the name has the form pppllls :

- ppp : prefix MEF if the library is portable, or else system mnemonics,
- llll: name to identify the library,
- s : suffix S meaning "source".

yymmddhh date (year, month, day) and time (hours) of the version.

comment (not yet implemented) is to put one of the following keywords :

- ADD : library or member to add on the site,
- REP : library or member to replace,
- DEL : library or member to delete,
- OLD : library or member which has not been modified from the last version.

For instance, a tape including a source library MEFBIBAS (including itself the Fortran subroutines SPAA and SPAB), as also a source library MEFBIBBS (including itself the Fortran subroutines SPBA and SPBB), would look like this :

```
./BIBLIF77 ADD NAME=MEFBIBAS,SSI=86033010
C/MEMBRF77 ADD NAME=SPAA,SSI=86032710
      SUBROUTINE SPAA
      ...
      END
C/MEMBRF77 ADD NAME=SPAB,SSI=86032710
      SUBROUTINE SPAB
      ...
      END
./BIBLIF77 ADD NAME=MEFBIBAS,SSI=86033010
C/MEMBRF77 ADD NAME=SPBA,SSI=86032710
      SUBROUTINE SPBA
      ...
      END
C/MEMBRF77 ADD NAME=SPBB,SSI=86032710
      SUBROUTINE SPBB
      ...
      END
```

This structure is designed to be compatible with the IBM processor named IEBUPDTE. However, the user can take advantage of it by creating programs himself. For instance, here is a simplified version of the program DECIEB, which is written in Fortran 77 (the full version is made of two parts : a main program DECIEB in the library PPAL, and a subroutine DECIEW in SYST) :

```
      PROGRAM DECIEB
      CHARACTER NAMOLD*80,NAMNEW*80,LINE*80
C
C OPENING THE INPUT FILE
      PRINT *,'INPUT FILE ?'
      READ  *,NAMOLD
      PRINT *,' '
      OPEN (10,FILE=NAMOLD,FORM='FORMATTED',STATUS='OLD')
C
C LOOP FOR EACH LINE OF THE INPUT FILE
  100 READ (10,'(A)',END=900) LINE
C
C NORMAL LINE
      IF (LINE(1:7) .NE. './BIBLI') THEN
         WRITE (20,'(A)') LINE
         GOTO 100
      END IF
C
C SPECIAL LINE
      CLOSE (20)
      INAME = INDEX(LINE,'NAME=')
      IF (INAME.EQ.0) STOP 'ERROR 1'
      IEND = INAME + 5
  300 IEND = IEND + 1
      IF (IEND .GT. 80) STOP 'ERROR 2'
      IF (LINE(IEND:IEND) .EQ. ',') GOTO 400
      IF (LINE(IEND:IEND) .EQ. ' ') GOTO 400
      GOTO 300
  400 NAMNEW = LINE(INAME+5:IEND-1)
      PRINT *,NAMNEW
      OPEN (20,FILE=NAMNEW,FORM='FORMATTED',STATUS='NEW')
      GOTO 100
C
C END OF EXECUTION
  900 CLOSE (20)
      CLOSE (10)
      END
```

From a disk file, this program generates several files. The name of these files is the one which follows the keyword NAME in lines beginning with ./BIBLI. This program may be modified as the user wishes, for instance to read directly on a tape and not on a disk, to add a suffix to the names of the generated files, to translate these files into lowercase letters, to suppress spaces in the end of each line (that may save disk space), to split a library into several members (then replace ./BIBLI by C/MEMBR),...

After this splitting work, one gets the files detailed in the table below. In fact, in this table, prefix MEF and suffix S of each portable library identifier are omitted.

LIST OF MODULEF LIBRARIES : 1986 Version

Subroutines in portable Fortran 77 - general libraries

BIHM : Dirichlet problem for the biharmonic operator ;

CONV : interactive preparation of Modulef data ;

COSD : construction of data structures from cards (BDCL, FORC, MILI, MAIL, NDL1, COOR, TAE) ;

ELAS, ELA2, ELA3, ELCP : finite element libraries for linear elasticity in 2D, 3D, or shell and plate elements (ELAS calls ELA2, ELA3 and ELCP - see table below) ;

ELNL : finite element library for nonlinear elasticity (large deformations, plasticity) ;

EVOL : transient problems of first and second order in time ;

FLUI : modules for computation of flow for an incompressible and viscous fluid in 2D ;

MAGN : magnetic study of a wave guide ;

NOP2, NOP3, NOPO : two and three dimensional meshes using the NOPO D.S. (NOP2 and NOP3 call NOPO - see table below) ;

PLMA : problems in plasma physics ;

POBA : generation of the values of basis polynomials at the numerical integration points ;

RESO, RESR, RESD, RESB : solution of linear systems and computation of eigenvalues in REAL or DOUBLE PRECISION type (RESO calls RESR and RESD, which call RESB (base) - see table below) ;

SYST : environment for Modulef users ;

THER : finite element library for thermal problems ;

TRAC : plotting of curves, meshes, isovalues, sections ;

UTIL : basic utilities ;

UTSD : data structure utilities ;

VIS3 : 3-D graphics ;

ZZZZ : specifications of subroutines and functions that a user may provide when calling some modules.

- Subroutines in portable Fortran 77 - graphics interfaces

BNF3 : Benson/Fortran 3D interface ;
BNTK : Benson/Tektronix interface (Plot10 system) ;
HAF3 : subroutine PCARA for generating "hard" characters (Fortran 3D software) ;
HATK : subroutine PCARA for generating "hard" characters (Tektronix software) ;
SOBN : subroutine PCARA for generating "soft" characters (Benson software) ;

For instance, suppose one has Benson software. To use this software on Tektronix with "hard" characters, use the BNTK and HATK libraries simultaneously.

- Main programs in portable Fortran 77

PPAL : programs calling the usual modules, for interactive use ;
TEST : numerical tests (elasticity, fluid mechanics, thermal problems).

- Nonportable subroutines

Among the following librairies, we only provide those which can be used at the site of the addressee :

IBMSYSA : FLIPFLOP, automatic utility implementation, in IBM assembler (see appendix IBM) ;
IBMUTIA : basic utilities (assembler), IBM version ;
IBMUTIM : basic utilities (macro-instructions), IBM version ;
MULSYSF : commands (Fortran) for Multics system ;
MULSYSP : commands (PL/1) for Multics system ;
MULUTIF : basic utilities (Fortran), Multics version ;
MULUTIP : basic utilities (PL/1), Multics version ;
SIRUTIA : basic utilities (Assembler), SIRIS 7/8 version ;
UNIUTIA : basic utilities (Assembler), UNIVAC version.

- Portable libraries containing data or text

POBD : data read by POBA library ;
PROD : documentation for MODULEF procedures ;
SYSD : data read by SYST library
TESD : data read by TEST library.

5. WRITING, IF NECESSARY, THE NONPORTABLE UTILITIES (STEP 4).

The nonportable part of Modulef is as small as possible. It is completely written in Fortran 77, but may require a slight adaptation from one site to another one. On the tape, the Multics version is generally provided, as an example (MULUTIF library). It contains the IASCII function, the "INFO functions", and subroutines OUVRIR and TRUNIT. All these utilities, as also the graphics interface, are described below :

IASCII function

This function returns the ASCII code of a character. At the present time, it is only called by the subroutine STRSFT of the library VIS3, in order to use different characters fonts on a graphics screen.

If the computer codes characters internally in ASCII, it's enough to keep the Multics version :

```
      FUNCTION IASCII(CHARAC)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C PURPOSE :
C     TO RETURN THE ASCII CODE OF A CHARACTER
C
C INPUT PARAMETER :
C     CHARAC : CHARACTER TO BE CODED
C
C VERSION : BULL/MULTICS
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      CHARACTER*1 CHARAC
      IASCII = ICHAR(CHARAC)
      END
```

Otherwise, it is possible to use the intrinsic function INDEX ('ABCDEFG...', CHARAC) or conversion arrays.

INFO functions

The matter is the 4 functions IINFO (of integer type), RINFO (real), DINFO (double precision), and KINFO (character). With these functions, programs can get some general information (unit number of the printer, smallest positive floating number, user's name, ...) without using commons.

To adapt these functions, read the provided version (MULUTIF) with a text editor, and modify it slightly according to the comments which are inserted in every function. For instance, the IINFO function in MULUTIF looks like this :

```
      INTEGER FUNCTION IINFO(KEYWRD)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C PURPOSE
C     TO RETURN INFORMATION OF TYPE INTEGER
C     *** MULTICS VERSION ***
C
C INPUT PARAMETER :
C     KEYWRD  : KIND OF INFORMATION TO BE RETURNED, IN CLEAR
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      CHARACTER*(*) KEYWRD
C
C 1. PORTABLE PART
C ----------------
C
      IF (KEYWRD .EQ. ...) THEN
         ...
      ELSE IF (KEYWRD .EQ. ...) THEN
         ...
C
C 2. NONPORTABLE PART
C -------------------
C
C INITIAL UNIT NUMBER OF THE INPUT DEVICE
      ELSE IF (KEYWRD .EQ. 'LECTEUR INITIAL') THEN
         IINFO = 5
         ...
C
C ERROR
      ELSE
         WRITE (IMPRIM,*) 'IINFO : UNKNOWN KEYWORD : ', KEYWRD
         IINFO = 0
      END IF
      END
```

Here, part 1 must of course not be modified, but part 2 must be examined. In our example, you may change the initial unit number of the input device * : replace 5, which is valid on Multics and many other machines, by the appropriate value (e.g. 105 on SIRIS 8). The values returned by the IINFO functions may be obtained by a simple constant assignment, but also by a system call (to get the CPU time,...). If such a value is difficult to obtain on the host machine, give any value, for instance 'SMITH' for the user's name.

* Let's recall that the user can, during the execution, modify the input device number : he must then use the %LECTEUR command (see INRIA Technical Report n°34, April 1984).

Subroutine OUVRIR

Experience has shown us that the OPEN statement of Fortran 77 is not totally portable. Hence we defined subroutine OUVRIR, that may require modifications on a new site :

```
      SUBROUTINE OUVRIR(UNIT,FILE,SPEC,RECL,IOSTAT)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C PURPOSE :
C     TO OPEN A FILE
C     CONVENTIONS ARE THOSE OF FORTRAN 77 STANDARD, EXCEPT FOR THE
C     FOLLOWING EXTENSION : IF SPEC CONTAINS 'OLD' AND 'DIRECT',
C     AND IF RECL=0, THEN THE FILE IS OPENED WITH THE RECORD LENGTH
C     THAT WAS GIVEN WHEN IT WAS CREATED
C
C INPUT PARAMETERS :
C     UNIT   : UNIT NUMBER OF THE FILE
C     FILE   : NAME OF THE FILE
C              IF FILE=' ', OPEN WITHOUT FILE=... SPECIFIER
C     SPEC   : LIST OF SPECIFIERS   (E.G. 'OLD,UNFORMATTED')
C     RECL   : RECORD LENGTH
C              IF RECL=0, OPEN WITHOUT RECL=... SPECIFIER
C
C OUTPUT PARAMETER :
C     IOSTAT : INPUT/OUTPUT STATUS CODE
C              IOSTAT = 0 IF NO ERROR CONDITION EXISTS
C
C VERSION : BULL/MULTICS
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Subroutine TRUNIT

This subroutine, too, is not totally portable :

```
      SUBROUTINE TRUNIT(UNIT)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C PURPOSE :
C     TO FIND A FREE UNIT NUMBER, I.E. NOT CONNECTED TO ANY FILE
C     THIS SUBROUTINE OFTEN PRECEDES SUBROUTINE OUVRIR
C
C OUTPUT PARAMETER :
C     UNIT   : UNIT NUMBER OF THE FILE
C
C N.B.
C     IF THIS SUBROUTINE DOES NOT WORK ON YOUR OPERATING SYSTEM,
C     YOU MAY USE A SIMPLIFIED VERSION :
C     WRITE (IINFO('I'),*) 'UNIT NUMBER ?'
C     READ (NF,*) UNIT
C
C VERSION : BULL/MULTICS
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Graphics interface

Modulef drawing modules call utilities that are normally designed for Benson drawing machines [1]. However, these utilities may be re-programmed for any graphics system. In particular, we provide on the tape interfaces for the Tektronix/Plot 10 system [2] (libraries BNTK and HATK) and for the Fortran 3D software [3] (libraries BNF3 and HAF3).

We describe below the specifications of the Benson utilities and then, as an example, Plot 10 utilities that are called by BNTK and HATK libraries. For more details, please consult :

[1] Benson, Programmes de Base, Niveau 1.
[2] Tektronix, Plot 10 Terminal Control System, User's Reference Guide.
[3] Groupe Graphique INRIA, Manuel d'Utilisation du Logiciel Graphique Fortran 3D, Modulef Report n°47.

a) Specifications of the Benson utilities

First of all, a few notions must be known :
- absolute mode : pen moves are relative to a fix origin.
- relative mode : pen moves are relative to the last position of the pen.
- object space : moves in centimeters.
- subject space : moves in units that are specific to the user's problem (see subroutines ECHEL and CVSB).

Let's describe now the different used subroutines :

CVSB (XS, YS, XA, YA, J)

converts subject coordinates (XS,YS) into object coordinates (XA,YA).

- If J = 0 or 1, XA = (XS*EX) + XO and YA = (YS*EY) + YO (absolute)
- If J = 2 or 3, XA = XS*EX and YA = YS*EY (relative)

ECHEL (EX,EY,XO,YO)

defines the subject space relatively to the last chosen object origin.

EX, EY : scales on X and Y axis, in number of centimeters per user's unit.

XO, YO : coordinates in centimeters of the subject origin relative to the object origin.

IBENA (IBUF, LBUF, ND)

initializes variables and opens the output file. Object and subject spaces are then identical, and the origin is set to the down left corner. The arguments, which are often useless on a graphics screen, are :

IBUF : work array

LBUF : length of the array

ND : output unit number

Call example : DIMENSION IBUF (1000)
CALL IBENA (IBUF, 1000, 10)

NOMBA (X,Y,J,FNP,NC,HX,HY,COSA,SINA)

draws a floating number :

X,Y,J : see PCARA

FNP : floating number to be drawn

NC ≥ 0 : number of decimal digits after the point (format F)

= -1 : no decimal digit, no point (format I)

< -1 : |NC| represents the number of significant digits (format E)

HX,HY,COSA,SINA : see PCARA

NOMBS (X,Y,J,FNP,NC,HX,HY,COSA,SINA)

identical with NOMBA, but (X,Y) in the subject space.

PCARA (X,Y,J,NLIS,NC,HX,HY,COSA,SINA)

draws a character string.

X,Y,J : define the starting point of the drawing (see TRAA)

NLIS : character string

NC : number of characters

HX,HY : width and height of a character, in centimeters.

COSA, SINA : sine and cosine of the angle between the direction of text and the X axis.

PCARS (X,Y,J,NLIS,NC,HX,HY,COSA,SINA)

identical with PCARA, but (X,Y) in the subject space.

PLUMA (K)

changes the pen, hence the drawing colors (K = 0,1,2,3).

PNUMA (XF,YF,NBLOC,XA,YA)

flushes the drawing buffer, sets the pen at position (XF,YF), closes the output file, and defines a new origin at (XA,YA). NBLOC, which is useless on a graphics screen, is a block number located by the Benson magnetic tapes readers.

TRAA (X,Y,J)

moves the pen to the point (X,Y). Depending on J, the move mode is absolute or relative, and the pen is up (no drawing) or down (drawing a segment) :

J	mode	pen
0	absolute	up
1	absolute	down
2	relative	up
3	relative	down

TRAS (X,Y,J)

identical with TRAA, but (X,Y) in the subject space.

b) Plot 10 utilities called by BNTK and HATK

ANMODE

puts terminal in alphanumeric mode.

AOUTST (NCHAR, IARRAY)

outputs first NCHAR characters from IARRAY.

CHRSIZ (ICHAR)

sets characters size to ICHAR value of 1,2,3 or 4 (1 = largest, 4 = smallest).

DRWABS (IX,IY)

draws line to screen coordinates IX,IY.

FINITT (IX,IY)

moves cursor to screen coordinates IX,IY and puts terminal in alphanumeric mode.

INITT (IBAUD)

initializes terminal, where IBAUD = transmission rate in characters per second.

FUNCTION KCM (RC)

converts RC centimeters to length in screen units.

MOVABS (IX,IY)

moves the beam to screen coordinates IX,IY.

TERM (ITERM, ISCAL)

identifies terminal type (ITERM = 0 to 3) and number of addressable points (ISCAL = 1024 or 4096).

6. COMPILING AND LINKING (STEPS 5 AND 6)

Source files must obviously be compiled. After that, the linker must generally be used to make "pre-linkings". The resulting files, when they are linked with a compiled main program, give then an executable load module.

Some particularities about Bull/Multics systems (dynamic linking), CDC, IBM... can be found in the appendices.

On most of the sites, all the libraries are separate (one source file, one compiled file, and one linked file per library).

Nevertheless, in some cases, it is better to group some linked libraries to make only one (the source libraries remaining separate to ease maintenance). This can sometimes reduce the CPU time and the number of inputs/outputs during linking, and at the same time ease the user's task. Besides, that grouping must absolutely be done for some linkers, which accept only a small number of libraries.

On the other hand, some machines set limits to the size of the resulting file, or the number of members per library.

The person being in charge of the implementation must therefore choose the best configuration for his or her own site. For instance, the following possibilities may be considered :

- to group all the linked libraries, except the interface libraries which contain necessarily subroutines with a same name (ibena, pcara, pnuma, ...),
- to group only the most used libraries, keeping the others separate.

Besides, on some systems, the result of the linking depends on the order in which the libraries are given (it is the case on Apollo, CDC, Cray 1, ...). The ideal order (if not compulsory) is such that the linker only reads each library one time to resolve the external references (search for missing subroutines). For that purpose, Modulef libraries are ordered thanks to the program PROIMP (see paragraph 7, command BB).

To realize this sort, we first consider the calls from one library to another, yielding the following matrix :

```
                    1         2         3
           12345678901234567890123456789012
 1 BIHM    *.2.............5626....*..*.4.
 2 CONV    .**1.1.11..15*...4.*....***.**.2
 3 COSD    ..*..*1..2.............*.*9.*.62
 4 ELA2    ...*.....................*..*.*.
 5 ELA3    ....*....................*..*.9.
 6 ELAS    ...****.................*1.*...
 7 ELCP    ...3..*.................*..*.*.
 8 ELNL    ..511..*.........6*33....*5.*.59
 9 EVOL    ..1.....*.......2.**..1.*..*...
10 FLUI    .........*........287..1.*3.*.8*
11 MAGN    ..2.......6........3.....*..*...
12 NOP2    ...........*.*..........2**.*..1
13 NOP3    ............**..........3**.*1.1
14 NOPO    ............*...........*5.*...
15 PLMA    .............*.........32....85
16 POBA    ..............*.........*1.1...
17 PPAL    .*.............1.....4...*3...1.
18 RESB    ................*.......6..*...
19 RESD    .................9*......*1.*.1.
20 RESO    .................6***....*..*.42
21 RESR    .................9..*....*1.*.1.
22 SYST    ....................*...45.....
23 TEST    ..*..4..42.54....*.*.....*..*...
24 THER    ......................*.*1.*.*.
25 TRAC    ..2..1...1.............**5.*.8*
26 UTIL    ........................**9...2
27 UTIL.FOR ........................69...1
28 UTIL.PL1 .............................5
29 UTSD    ........................**2*..3
30 VIS3    ........................**.**.*
31 ZZZZ    ........................*3.....
32 ????    .........................
```

Legend :

.	M(I,J) = 0
1 to 9	M(I,J) = 1 to 9
*	M(I,J) ≥ 10

This matrix is built by incrementing M(I,J) each time a procedure of I calls a procedure of J. Now, let the relation "precedes", noted "<", defined by :

$$I < J \quad \text{iff } M(I,J) \neq 0 \text{ and } I \neq J$$

If < is a partial order relation (there must not be any circuit I<J<K ... <I), then we can order the libraries by a topological sort, with a result that is not necessarily unique. The Modulef libraries being designed to verify this partial order property, one of the possible results is the following :

```
BIHM   MAGN   PLMA   PPAL   TEST
CONV
NOP2   NOP3
ELNL   EVOL   TRAC
COSD
FLUI
ELAS   RESO
ELCP   RESD   RESR
ELA2   ELA3   NOPO   POBA   RESB   THER   VIS3
SYST   UTSD   ZZZZ
UTIL
Nonportable basic utilities
```

The previous table is constructed so that :

- if A < B, then A is on a line located over B,
- if two libraries A and B are located on the same line, they are independent : we have neither A < B, nor B < A (this is implied by the previous assertion),
- the libraries that are not called by any other library are all located on the first line,
- the libraries that do not call any other library are all located on the last line.

For instance, if libraries EVOL, NOP3, TRAC, UTIL, UTSD are necessary for the execution of a program, one can adopt the following order :

NOP3, EVOL, TRAC, UTSD, UTIL
or NOP3, TRAC, EVOL, UTSD, UTIL.

The linker looks for the missing subroutine first in NOP3. Then it looks for further references in EVOL (resp. TRAC), and so on... Finally, all the necessary references to the execution are resolved. For example, the indicated order guarantees that no subroutine of UTSD calls a subroutine of NOP3, EVOL, or TRAC.

7. CREATING THE PROCEDURES DATA BASE (STEP 7)

The user frequently needs an answer to the following questions :

- which procedures are in library L ?
- which procedures does procedure P call ?
- by which procedures is common C used ?
- which libraries are required to run procedures P1, P2, P3 ?
- etc...

For that reason, a network data base system has been developed. To create this data base, just execute program PROCAT in library PPAL. This program calls the following subroutines :

```
UTIL ARRET  | SYST CATA   | SYST CATALL |
SYST CATBFE | SYST CATBFL | SYST CATCHC | SYST CATCRE
SYST CATDEC | SYST CATERR | SYST CATFER | SYST CATLON
SYST CATOUV | SYST CATRAM | SYST CATREC | SYST CATVDL
SYST CATX   | SYST CATXL  |             |
UTI. DINFO  |             | UTIL ICHAR4 | UTI. IINFO
            | UTIL INITIS | UTI. KINFO  | UTIL LECTEU
UTIL LIBACT | UTIL LIBANA | UTIL LIBARG | UTIL LIBCAL
UTIL LIBCAR | UTIL LIBCOM | UTIL LIBDCL | UTIL LIBENT
UTIL LIBLEC | UTIL LIBNMB | UTIL LIBRGL | UTIL LIBSCN
UTIL LIBTBK | UTI. OUVRIR |             |
UTIL PRALDY | PPAL PROCAT | SYST PROCAW | SYST PROCB
SYST PROCB1 | SYST PROFIC | SYST PRORES |
UTI. TRUNIT |             |             |
```

For machines with a static linker, the required libraries are PPAL, SYST, UTIL and finally the nonportable utilities library (or libraries : for instance, on IBM, IBMUTIA and IBMUTIF).

In other respects, program PROCAT :

- reads data that are in member SYSD of library MEFSYSDS,
- creates a direct access file through an OPEN statement in subroutine CATOUV (RECL = 1024 words = 4096 bytes on IBM). The name of this file is returned by function PROFIC and may be modified :

```
      CHARACTER*32 FUNCTION PROFIC(I)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C PURPOSE : TO RETURN THE NAME OF THE FILE WHICH IS CREATED BY PROCAT
C           AND READ BY PROIMP
C ***
C *** NON PORTABLE FUNCTION :
C *** RETURN A FILE NAME WHICH IS VALID FOR THE SYSTEM :
C *** 'PRO.CAT', 'PROCAT', '/UDD/MODULEF/PROCAT.DIR', ...
C ***
C INPUT PARAMETER :
C     I : UNUSED (BUT COMPULSORY IN FORTRAN)
C+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      PROFIC = 'PRO.CAT'
      END
```

We get the following on Multics :

```
ac x >udd>Modulef>v>s>syst.indep.data.s sysd.data
r 17:29 0.129 26

ted -pn sysd.data
1
A GENE C 860323 16072749 ; A NMPR C A2DPD A2DPR...
$
2 12 3 2 1 1 104 34 5 17 48 30 1 3 36 17 8 ; F F
q
r 17:29 0.195 30
```

```
procat
 NOM DU FICHIER DE DONNEES ?
sysd.data
 NOMBRE DE MOTS DANS GENE : 4
 NOMBRE DE MOTS DANS NMPR : 5472
 NOMBRE DE MOTS DANS NMBI : 64
 NOMBRE DE MOTS DANS NMCO : 422
 NOMBRE DE MOTS DANS PRBI : 2736
 NOMBRE DE MOTS DANS PRAP : 23884
 NOMBRE DE MOTS DANS PRCO : 8969
 NOMBRE DE MOTS DANS LGCO : 211
 NOMBRE DE MOTS DANS BIBI : 1024
r 17:31 70.938 47
```

After creating the direct access file, the data base is queried through program PROIMP. This program calls the following subroutines :

SYST CATALL	SYST CATBFE	SYST CATBFL	SYST CATCHC
SYST CATDEC	SYST CATERR	SYST CATFER	SYST CATLON
SYST CATOUV	SYST CATRAM	SYST CATREC	SYST CATVDL
SYST CATX	SYST CATXL		UTI. IINFO
	UTI. OUVRIR	SYST PROBB	SYST PROBBA
SYST PROBBC	SYST PROBBM	SYST PROBBP	SYST PROBBT
SYST PROFER	SYST PROFIC	SYST PROIMA	SYST PROIMN
PPAL PROIMP	SYST PROIMT	SYST PROIMW	SYST PROLC
SYST PROPB	SYST PROPC	SYST PROPC1	SYST PROPC2
SYST PROREC			

During the execution, program PROIMP reads the file created by program PROCAT, the name of which being returned by function PROFIC. The user is continuously guided by menus, as shown in the following example (the — sign precedes the explanations that have been added here) :

```
PROIMP

**** COMMANDE ?                    — Asks for the next command
?                                  — Brief list of the available commands
? ?? BB G LC NB NC NP PA PB PC P* Q

**** COMMANDE ?
??                                 — Long list of the available commands

?    AIDE RESUMEE
??   AIDE DETAILLEE
BB   RELATIONS DE BIBLIOTHEQUES A BIBLIOTHEQUES
G    GENERALITES
LC   LONGUEUR D'UN COMMON
NB   NOMS DES BIBLIOTHEQUES
NC   NOMS DES COMMONS
NP   NOMS DES PROCEDURES
PA   RELATIONS DE PROCEDURES A APPELEES
PB   RELATIONS DE PROCEDURES A BIBLIOTHEQUES
PC   RELATIONS DE PROCEDURES A COMMONS
P*   FERMETURE TRANSITIVE DES PROCEDURES APPELEES
Q    QUITTER LE PROGRAMME (STOP)
```

```
**** COMMANDE ?
P*

CHOISISSEZ LE MODE D'UTILISATION :
   A          ARBRE DES PROCEDURES
   B          TABLEAU DES BIBLIOTHEQUES
   P          TABLEAU DES PROCEDURES
   BP         TABLEAU DES BIBLIOTHEQUES + PROCEDURES
   (RETURN)   QUITTER
A

PROCEDURE ?
READRE                          — Tree of the subr. called by READRE :
   1 ADRESS   CHAR4             — READRE calls ADRESS, TROUVE, TUER
   2 '        DECAL    TYPMOT   — ADRESS calls CHAR4, DECAL, ICHAR4,
   3 '        ICHAR4            —                IMATAB, TUER, TYPMOT
   4 '        IMATAB            — etc...
   5 '        TUER     ICHAR4   (   3)
   6 '        TYPMOT   (   2)
   7 TROUVE   ICHAR4   (   3)   — Each number in parentheses refer to the line
   8 TUER     (   5)            — where the subr. has already been mentioned

PROCEDURE ?


CHOISISSEZ LE MODE D'UTILISATION :
   ...
BP

PROCEDURE ?
RSTSDE

PROCEDURE ?
SAUSDS

PROCEDURE ?          — Procedures called by RSTSDE and SAUSDS,
                     — preceded by the libraries that include them

BIBLIOTHEQUES + PROCEDURES :

UTIL     ADRESS   UTIL     CHAR4    UTIL     DECAL    UTIL     ICHAR4
UTIL     IMATAB   UTIL     INCANO   UTIL     INCAPA   UTSD     INICSD
UTIL     NUMALP   UTIL     READRE   UTSD     RSTSDE   UTSD     SAUCSD
UTSD     SAUSDS   UTIL     TROUVE   UTIL     TRTATA   UTIL     TUER
UTIL     TYPMOT

NOMBRE D'ELEMENTS DU TABLEAU PRECEDENT :      34

CHOISISSEZ LE MODE D'UTILISATION :
   ...


**** COMMANDE ?
Q
 NOMBRE DE MOTS UTILISES : 35130 /40000

STOP
r 18:23 2.870 60
```

8. GENERATING THE POBA FILE (STEP 8)

Some Modulef subroutines (computation of the elementary arrays : mass or stiffness matrix, right hand side, ...) may read numerical values on a particular file, called POBA, which must be generated once for all when implementing. To get this file, the main program POBAXX of library PPAL must be run. This program calls the following subroutines (table obtained with PROIMP, see paragraphe 7) :

```
UTIL ADRESS | UTIL ARRET  | UTIL AZEROD | UTIL CHAINT
UTIL CHAR4  |             |             | UTIL DEFDIR
UTI. DINFO  |             | UTIL ECPDIR | UTIL ECRDIR
UTSD ESTASF | UTIL FERDIR | UTIL ICHAR4 | UTI. IINFO
            | UTIL IMALDY | UTIL INITIS | UTIL INTCHA
UTI. KINFO  |             | UTIL LECDIR | UTIL LECTEU
UTIL LECTUT | UTIL LEPDIR | UTIL LIBACT | UTIL LIBANA
UTIL LIBARG | UTIL LIBCAL | UTIL LIBCAR | UTIL LIBCDP
UTIL LIBCOM | UTIL LIBCSP | UTIL LIBDCL | UTIL LIBENT
UTIL LIBERE | UTIL LIBLEC | UTIL LIBLOG | UTIL LIBNMB
UTIL LIBRDP | UTIL LIBRGL | UTIL LIBRSP | UTIL LIBSCN
UTIL LIBTBK | UTIL MESDIR | UTIL NBMCHA | UTIL OUVDIR
UTI. OUVRIR |             |             | UTIL PN2DDE
UTIL PN2DIN | UTIL PN2DPR | UTIL PN2DVA | UTIL PN2RDE
UTIL PN2RVA | UTIL PN3DDE | UTIL PN3DVA | UTIL PN3RDE
UTIL PN3RVA | POBA POBA   | PPAL POBAXX | UTIL PRALDY
POBA PROD   | POBA TA2P3D | POBA TA2Q2S | POBA TA3COQ
POBA TA3Q2D | POBA TA3Q2S |             | UTIL TRCHTA
UTIL TRTATA | UTI. TRUNIT | UTIL TUER   |
POBA VAD2Q5 | POBA VAD3Q1 | POBA VADAP1 | POBA VADAP2
POBA VADAQ1 | POBA VAR2Q4 | POBA VAR2Q5 | POBA VDQ2R2
POBA VDQ2R3 | POBA VPD1   | POBA VSQ2R2 | POBA VSQ2R3
UTIL ZALDY1 | UTIL ZALDY2 | UTIL ZALDY3 | UTIL ZALDY4
UTIL ZALDY6 |             |             |
```

For machines with a static linker, the required libraries are PPAL, POBA, UTSD, UTIL and finally the nonportable utilities library (or libraries : for instance, on IBM, IBMUTIA and IBMUTIF).

In other respects, program POBAXX :

- reads data that are in member POBD of library MEFPOBDS,
- creates a direct access file POBA.

We get the following listing on Multics :

```
ac x >udd>Modulef>v>s>poba.indep.data.s pobd.data
r 16:56 0.110 3

ted -pn pobd.data
1,3
$     HEXAEDRE Q2 $
 $ IP LA PERMUTATION DE COMPOSANTE PAR COMPOSANTE A NOEUD PAR NOEUD
      1 4 7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52
$-2,$
  0.27845619121185239D+01 -0.40514602929382537D+00
  0.40514602929382537D+00  0.40514602929382537D+00
 -0.40514602929382537D+00  0.27845619121185239D+01
q
r 16:57 0.158 7
```

```
pobaxx
NOM DU FICHIER DE DONNEES ?
pobd.data
NOM DU FICHIER POBA ?
poba.direct

LE TABLEAU 3Q2I  D'ADRESSE      1 A    134 MOTS  SA PREMIERE PAGE EST      2
     1     4     7    10    13    16    19    22    25    28    31    34
    37    40    43    46    49    52    55    58     2     5     8    11
    14    17    20    23    26    29    32    35    38    41    44    47
    50    53    56    59     3     6     9    12    15    18    21    24
    27    30    33    36    39    42    45    48    51    54    57    60
    27     9
   .17147        .27435        .17147        .27435        .43896        .27435
  .17147        .27435        .17147        .27435        .43896        .27435
  .43896        .70233        .43896        .27435        .43896        .27435
  .17147        .27435        .17147        .27435        .43896        .27435
  .17147        .27435        .17147        .30864        .49383        .30864
  .49383        .79012        .49383        .30864        .49383        .30864

                .......................

FIN NORMALE ( AAPOBA ) :

                NOMBRE DE TABLEAUX  (NBTAST) :     44
                NOMBRE DE PAGES     (NODEPA) :    191
                DESCRIPTION DES TABLEAUX :
                ---------------------------

3Q2I      0  134    2     3Q2P      0 1080    3     3Q1P      0  128    8
3Q2Q      0 3240    9     3Q2V      0 1080   22     3Q1V      0  432   27
3Q2D      0 3240   29     2Q2V      0  144   42     2Q2D      0  288   43
Q2Q2      011340   45     3Q1D      0 1296   90     2Q1D      0  144   96
3Q1Q      0  384   97     3T2I      0   38   99     3T2P      0  540  100
3T1P      0   64  103     3T2Q      0 1620  104     3T2V      0  540  111
3T2D      0 1620  114     2T2V      0   72  121     2T2D      0  144  122
T2Q2      0 5670  123     3T1D      0  648  146     2T1D      0   72  149
3T1Q      0  192  150     3T2F      0 1080  151     3T1F      0  432  156
2P25      0  268  158     3COQ      0 1458  160     2P13      0   22  166
2Q13      0  106  167     2P3D      0  758  168     2Q25      0  452  171
2T25      0  227  173     3Q1G      0  632  174     2P26      0  268  177
POIS      0   26  179     PP2S      0  936  180     VP2S      0  208  184
DP2S      0  416  185     PB2S      0   30  187     DB2S      0   30  188
DPQ1      0  208  189     2T30      0  254  190     VIDE      0    0    0
VIDE      0    0    0     VIDE      0    0    0     VIDE      0    0    0
VIDE      0    0    0     VIDE      0    0    0     VIDE      0    0    0
VIDE      0    0    0     VIDE      0    0    0     VIDE      0    0    0
VIDE      0    0    0     VIDE      0    0    0     VIDE      0    0    0
VIDE      0    0    0     VIDE      0    0    0     VIDE      0    0    0
VIDE      0    0    0     VIDE      0    0    0     VIDE      0    0    0

STOP
r 16:58 71.157 55
```

9. TESTS AND NORMAL USE (STEPS 9 AND 10)

After implementing the Modulef libraries, we strongly advise to run the test programs, which are in library TEST (source) and TESD (data), and documented in a separate notice (n°87). You may at this time detect implementation errors, and this gives you some use examples.

After that, about every two monthes, updatings are delivered for possible error corrections, improvements and extensions. For everyone's interest, please tell us if you notice any trouble.

Multics tapes contain mainly source, object and bound libraries, which avoids the initial splitting task required by multicomputer tapes.

The tape is written by the backup_dump processor. It may be restored by backup_load or retrieve. The usage of these commands is described in Multics manual AM 81, Multics Operator's Handbook.

The execution of a **dumping** looks like this :

```
backup_dump >udd>Modulef>v -map -debug -error_on
Type primary_dump_tape label: m00562,den=6250
Mounting tape m00562 for writing
Mounted tape m00562 on drive tapb_06
Begin at 05/06/81  1719.6 hfe Wed
Map attached to file "!BBBJLCzjGHQfXP.backup.map".
Normal termination 05/06/81  1732.8 hfe Wed
r 17:32 54.156 7728
```

The file !BBB...backup.map contains a list of the processed segments and directories. A listing of it is attached to the tape. To give an example, a **complete retrieval** from the tape could be done this way :

```
backup_load -debug -noquota -nosetlvid -notrim -error_on
Input tape label: m00562,den=6250
Mounting tape m00562 for reading
Mounted tape m00562 on drive tapa_01
Begin at 04/30/82  1238.9 hfe Fri
Map attached to file "!BBBJLxHwmkJQjP.backup.map".
End of reel encountered.
bk_input: Are there any more tapes to be reloaded? no
Normal termination 04/30/82  1247.1 hfe Fri.
r 12:47 53.758 6135
```

It is generally more convenient to use the retrieve command, which is used to **retrieve** files in **any directory**, or to **retrieve only a part** of these files.

For example, to retrieve in the directory >udd>Project>modulef>s the files that are on the tape in >udd>Modulef>v>s, the user may :

- create a file, named ret, containing the line :

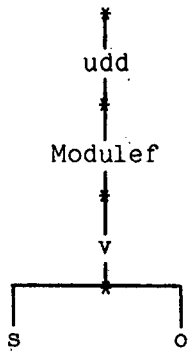
```
>udd>Modulef>v>s>** = >udd>Project>modulef>s
```

- call retrieve :

```
retrieve ret -debug -noquota -nosetlvid -notrim -error_on
Input tape label: M01152,den=6250
Mounting tape M01152 for reading
Mounted tape M01152 on drive tapa_01
Begin at 11/14/84  1037.3 hfh Wed
End of reel encountered.
retrieve: Are there any more tapes to be reloaded? no
Normal termination 11/4/84  1041.0 hfh Wed.
r 10:41 28.271 1901
```

Implementing the tape

In 1600 bpi, the tape contains almost only source files. These files must be compiled and bound. On the other hand, in 6 250 bpi, a complete version is provided. The configuration used at INRIA, which may be transcribed on other sites, is the following :



The directory v (current version, which is delivered) contains the bound libraries (hence executable) and two sub-directories s (containing source files) and o (containing object files).

The implementor of a 1600 bpi tape must first read the files of the tape into the directory >udd>Modulef>v. The names of these files are of the form :

comp1.comp2.comp3.s.archive

with

comp1 : name of the library (4 characters),

comp2 : kind of computer on which the library may be used ("indep" : machine-independent library),

comp3 : language used.

This way of naming is different from that of multicomputer tapes, for in this latter case the identifier length is limited to 7 characters (8 with the suffix "S"). The translation from a Multics identifier comp1.comp2.comp3 to a multicomputer identifier libnam is made the following way :

If comp2.comp3 = "indep.fortran", then libnam = MEFcomp1

Else libnam = first 3 characters of comp2 (or MEF if comp2 = "indep"), followed by the first 3 characters of comp1, followed by the first character of comp3.

For instance, conv.indep.fortran becomes MEFCONV, poba.indep.data becomes MEFPOBD, and util.multics.fortran becomes MULUTIF.

The source and object files can be handled by the Multics command "archive" or "ac" (AG 92, Commands and Active Functions).

At INRIA, compiling and binding the archive reso.indep.fortran.s.archive would be made as below :

```
cwd >udd>Modulef>v>s
ac x reso.indep.fortran.s
ft -ansi77 -card -no_auto_zero -ntb ((segs *.fortran))
dl *.fortran
ac ad <o>reso.indep.fortran (segs *)
cwd <
bd o>reso.indep.fortran >udd>Modulef>bind
```

where >udd>Modulef>bind.archive is an archive with only one member xxxx.bind containing the following lines :

```
Global : no_link ;
Addname ;
```

So, from the source archive v>s>reso.indep.fortran.s.archive, we created the object archive v>o>reso.indep.fortran.archive and the executable segment v>reso.indep.fortran (this library is named RESO in the "list of Modulef libraries").

There is a slight difference for interface libraries, which include necessarilly subroutines with a same name (ibena, pcara, pnuma, ...) and which must be called when the user asks explicitly for it. Then, for each interface library, a separate directory (named for instance bnf3, bntk, haf3,...), may be created at the same level than directory v.

Then, to run a Modulef program, the user just needs to add directory >udd>Modulef>v in the search rules ("asr" command), and if required some directories for interface libraries.

For example, after implementing the required libraries, the POBA file can be generated as described in paragraph 8. On 6250 bpi tapes, this file has the absolute pathname >udd>Modulef>v>donnees>poba.direct, and it must have a read access for every Modulef user. This directory contains other data files, in particular PRO.CAT (see paragraph 7).

The implementation being achieved, the test programs provided in test.indep.fortran.s.archive can now be executed. Note that some members contain at the same time a main program and a function, which implies to give several names to the compiled segment ("an" command).

Some modules may cause floating-point underflows. To avoid this, just type before executing (if this is not already done in the start_up) :

```
exponent_control -rt underflow
```

APPENDIX : IMPLEMENTING ON CDC

Reading a Modulef tape with block-length 5120, ASCII, 1600 bpi :

```
/JOB IMPLEM
IMPLEM,T100.
USER,XXXXXXX,XXXXXX.
CHARGE,XXXX,XXXXXX.
PURGE,MEF/NA.
DEFINE,MEF.
LABEL,CARTE,D=PE,CV=AS,NT,PO=R,F=S,LB=KU.VSN=M00237
COPY,CARTE,MEF,,C1.
DAYFILE,JOUR.
REPLACE,JOUR.
EXIT.
DAYFILE,JOUR.
REPLACE,JOUR.
/EOF
```

Then, we get a file with 640 characters per block (with colons between records). The following program reads this file with 5120 characters blocks, and creates a source file that may be compiled (fixed blocked 80 characters) :

```
      PROGRAM PCAR(TAPE1,TAPE2)
      DIMENSION AA(512),B(8)
    1 BUFFER IN (1,1) (AA(1),AA(512))
      X = UNIT(1)
      DO 20 K = 1,64
      DO 10 I = 1,8
      B(I) = AA(I+8*K-8)
   10 CONTINUE
      WRITE (2,1000) (B(L),L=1,8)
 1000 FORMAT (8A10)
   20 CONTINUE
      IF (X) 1,2,2
    2 STOP
      END
```

Writing a Modulef tape with block-length 5120, ASCII, 1600 bpi :

```
/JOB IMPLEM
IMPLEM,T100.
USER,XXXXXXX,XXXXXX.
CHARGE,XXXX,XXXXXX.
ATTACH,OMULTIC/NA.
LABEL,CARTE,D=PE,CV=AS,NT,PO=W,F=S,LB=KU.VSN=M00237
COPY,OMULTIC,CARTE,,C2,TC=I,CC=5120,PO=R.
DAYFILE,JOUR.
REPLACE,JOUR.
EXIT.
DAYFILE,JOUR.
REPLACE,JOUR.
/EOF
```

APPENDIX : IMPLEMENTING ON IBM

1. Implementation utilities

FLIPFLOP utility

For every record beginning with a "/" in column 2 and such that the character string beginning in column 3 is equal to the one passed as an argument, this utility changes every "." of column 1 into "C" and vice versa.

N.B. If the argument field is empty, the only thing considered is the "/".

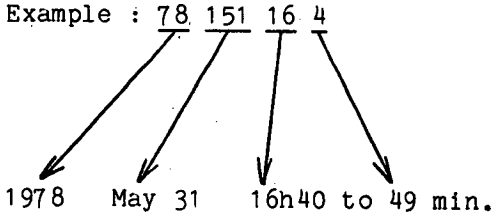
UPDATE utility

This program has the same functionality than the OS utility IEBUPDTE but, in addition, the date and time of the last processing (ADD, REPL, CHANGE or REPRO) is automatically recorded in the SSI (Status System Information) area of the directory.

The date and time coding is of the form yydddhhm :

- yy year,
- ddd day number in the year,
- hh hours,
- m first digit of the minutes number.

Example : 78 151 16 4



N.B. A work file SYSIN 2 is required.

CUTTER utility

This program, patched on the FORTRAN compiler IFEAAB level 2.2 sept. 76, uses as input the files SYSTERM, SYSLIN and SYSPRINT of the compiler and inserts a separating record before each module. The separating records are, depending on the option that is passed as an argument :

$$./ \left\{ \begin{matrix} \text{ADD} \\ \text{REPL} \end{matrix} \right\} \text{NAME} = \text{module name}$$

for the source file and

NAME module name [(R)]

for the object file.

N.B. If the argument field is empty or if an option different from ADD or REPL is passed, then ADD is taken.

LISTPDS utility

This program is not used for implementing but, as it is on the tape in library IBMUTDAS, it is worth knowing its abilities.

LISTPDS makes a selective copy of the contents of a library into different output files.

Output file SYSPRINT ; it includes, for every selected member :
- member name, date and time of the last updating,
- contents of the member (record length 133).

Optional output files : if the corresponding DD cards are omitted, these files are ignored.
- SYSPUNCH : contents of the selected members (record length 80)
- $\left\{ \begin{matrix} \text{ADD} \\ \text{REPL} \\ \text{CHANGE} \end{matrix} \right\}$: like SYSPUNCH but with separating records

$$./\text{UPDATE} \left\{ \begin{matrix} \text{ADD} \\ \text{REPL} \\ \text{CHANGE} \end{matrix} \right\} \text{NAME} = \text{member name, SSI} = \text{date}$$

- SYSLIST : contents of the directory

Selecting members

According to a list of keys passed as an argument, members are selected by the beginning or all the characters of their name, or by the last date of updating.

To describe the argument field of LISTPDS, we use the BNF notation below :

<argument field> ::= <name>| '<list of keys>'| <empty>

<name> ::= <letter>| <name><letter>| <name><digit> max 8

<list of keys> ::= <key>| <list of keys>,<key> max 100

<key> ::= <name selection key>| <date selection key>

<name selection key> ::= <name>| <name><blank>

<date selection key> ::= <relation operator> <date>

<relation operator> ::= <negation operator>
| <elementary relation operator>
| <relation operator> <elementary relation operator>

<negation operator> ::= ¬

<elementary relation operator> ::= < | = | >

<date> ::= <digit>|date><digit> max 8

N.B.

- The different elementary relation operators are combined together with a "or". Negation applies to the result of combining all the elementary relation operators that are after the negation operator.
- If there are several date selection keys, the last one only is considered.
- If the argument field is empty, all the members are selected.
- The different name selection keys are first combined together with a "or" before being combined with a "and" at the last date selection key.

Example :

```
//EXEC LISTPDS,LIBRARY='CCMT.P.MEFVISUS',
//             PARM='TEST,FR ,>=800422,CAL'
//SYSLIST DD SYSOUT=A
```

prints the contents of every member whose name begins with TEST or CAL as also the FR member, provided that these members have been updated since 31 may 1978 included. Contents of the directory are also listed.

2. DYNAMIC MEMORY ALLOCATION UTILITIES

2.1. DYNAM subroutine

This subroutine makes the dynamic memory allocation of one or more arrays (aligned on double words addresses) and calls the subroutine using them.

2.1.1. Compulsory arguments :

- number of arrays to allocate dynamically (positive integer)
- name of the subroutine for which these arrays are allocated (to declare as an EXTERNAL)
- sizes in bytes of the different arrays (positive integers)

2.1.2. Optional arguments :

All the other arguments that the subroutine may need.

2.1.3. Example

```
      EXTERNAL   MODULE
      N1  = 4Ø
      N21 = 1Ø
      N22 = 3
      N3  = 3Ø
      CALL DYNAM (3,MODULE , 4*N1,8* N21*N22, 16*N3,
     (N1, N21, N22, N3)
      STOP
      END
      SUBROUTINE MODULE (T1, T2, T3, N1, N21, N22, N3)
      INTEGER T1 (N1)
      REAL*8   T2 (N21, N22)
      COMPLEX*16 T3 (N3)
        ≀
      RETURN
      END
```

Effects :

- Dynamic allocation of three arrays containing respectively 4 x 40 = 160, 8 x 10 x 3 = 240, 16 x 30 = 480 bytes.

- Call to subroutine MODULE that gets three arrays T1, T2, T3 that are considered respectively as an integer vector, a real double precision matrix and a complex double precision vector, and the 4 optional arguments N1, N21, N22, N3.

- Execution of the subroutine.

- Deallocation of the dynamic arrays during RETURN.

Note : DYNAM being re-entrant, calls to DYNAM are possible in MODULE as well as in any subroutine that may be called by MODULE.

2.2. LAYOUT function

This function watches the virtual memory contents during the call and computes, taking into account files that are not yet used but that have been assigned, the number of available bytes in the desired REGION.

2.2.1. Argument

To be considered as a FUNCTION by the FORTRAN syntax analyser, LAYOUT must get at least one argument (that will be ignored).

2.2.2. Result

Available region in bytes.

Note : if the 3rd word NNN of COMMON TRAVAI is > 0, a detailed listing of the analysis is provided.

2.2.3. Optimal allocation of the super-array with LAYOUT and DYNAM

Example :

```
EXTERNAL  MODULE
COMMON/TRAVAI/IM, LM, NOMTAB, NNN, MI (1020)
NNN = 1
NARG = 2
LM = (NBMOTS(LAYOUT(LM))-(18+NARG+1)/2*2)/1024*1024
NNN=0
IF (LM.GT.Ø) CALL DYNAM (1, MODULE, NBMACH (LM),LM)
STOP
END
SUBROUTINE MODULE (M,N)
DIMENSION M(N)
  ≷
RETURN
END
```

Explanations :

- NBMOTS (LAYOUT (LM)) = number of available words.

- (18+NARG+1)/2 * 2 = number of words that are necessary to DYNAM to insure re-entry.

- 18 words for the registers "save area".

- as many words as arguments to pass to MODULE.

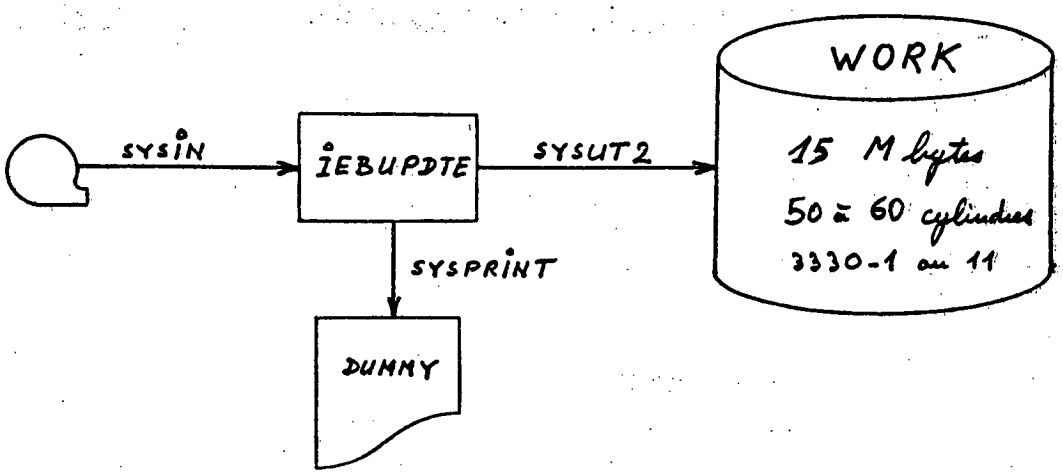
- all that is rounded to the nearest greater even number, for the super-array will be framed on a double-word address.

- the memory allocation being made by 4 Kbytes pages, it is necessary to round to the lower number of pages, so that the allowed region won't be overflown.

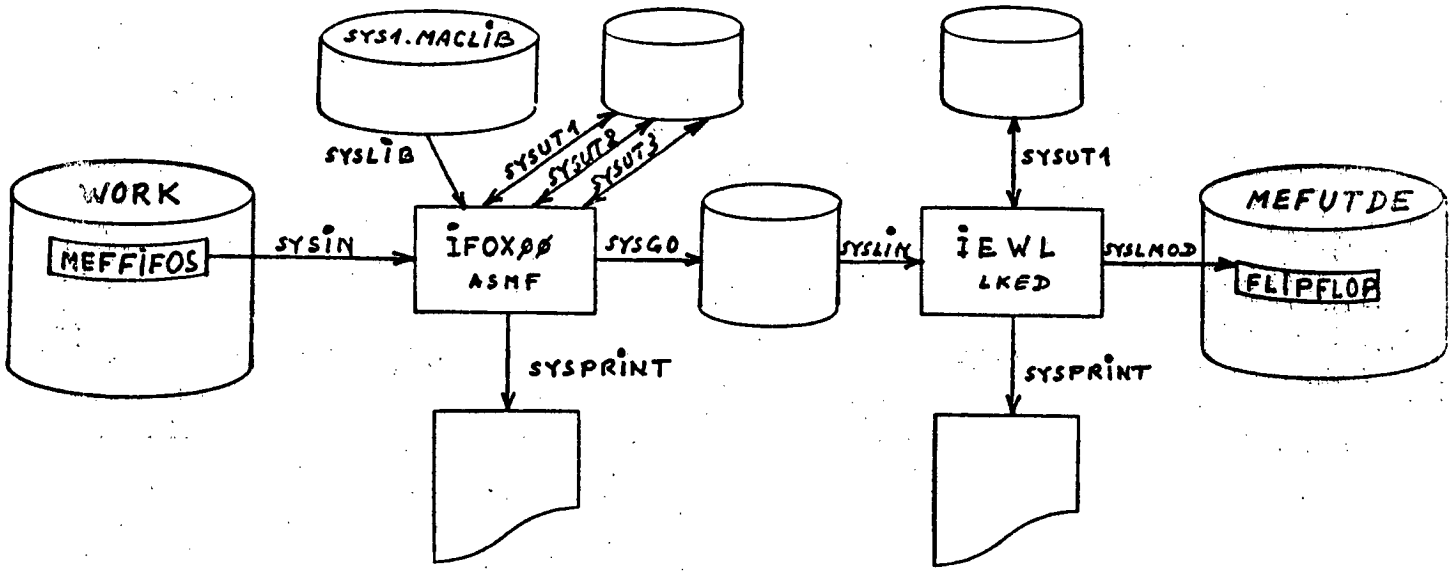
3. IMPLEMENTATION SCHEDULE

Each implementation step having to call only already implemented modules, this chronological order must absolutely be followed.

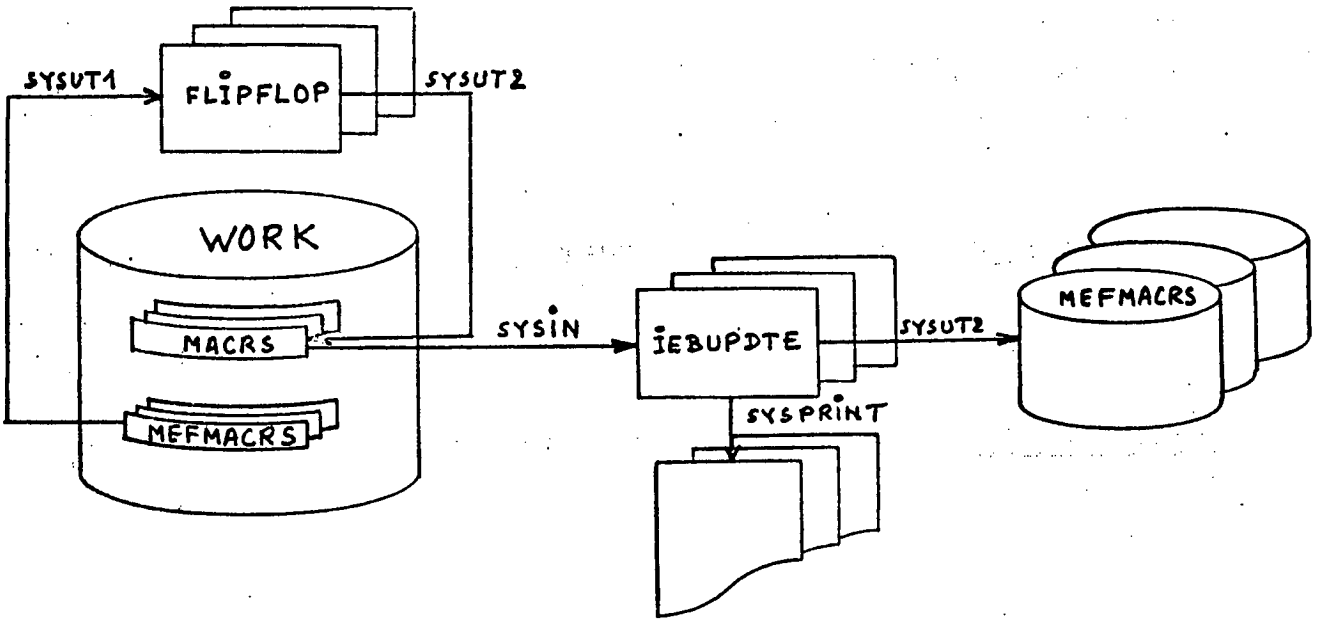
3.1. Tape reading and primary splitting in a library whose each member will contain all the records of a MODULEF library.



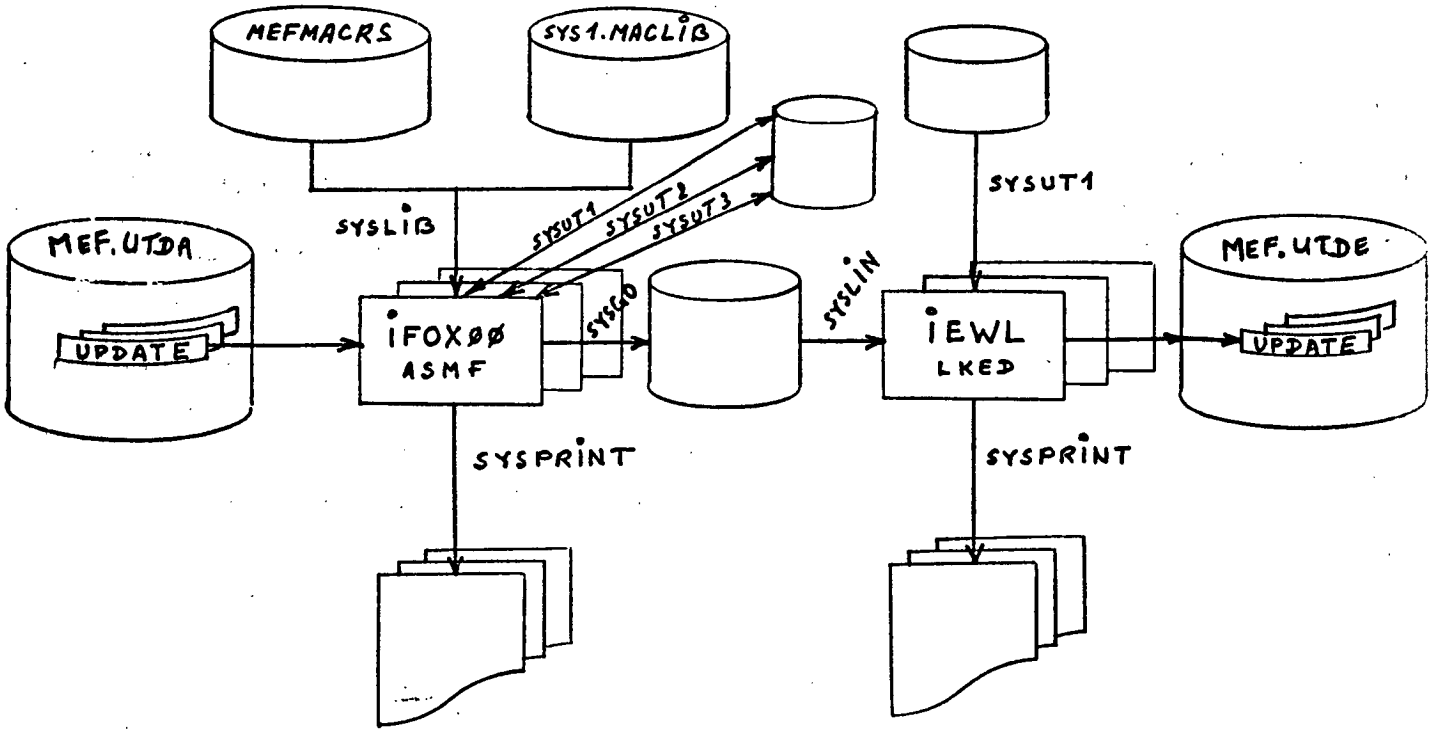
3.2. Assembling and linking the FLIPFLOP utility



3.3. Splitting all non FORTRAN library

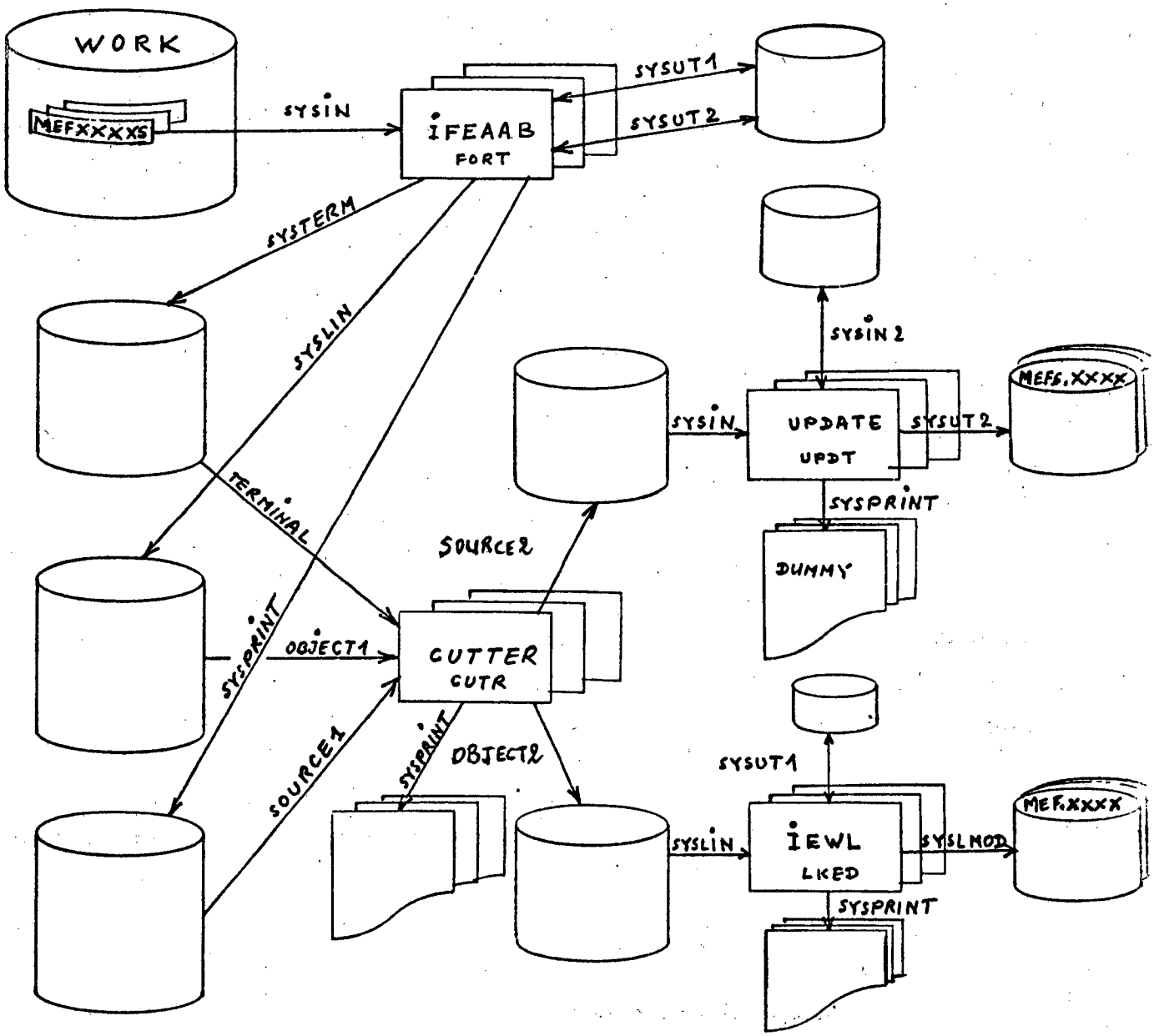


3.4. Assembling and linking all the assembly modules



3.5. Compiling and splitting, linking and splitting all the FORTRAN modules

The EXPAND procedure does in only one pass, owing to the CUTTER utility, compiling and linking all the modules of a library. With only 4 "steps", we get directly a source modules library and a compiled/linked module library. Time saving for allocating and deallocating ressources and for initializing the compiler and linker is considerable compared with the 2n "steps" that are needed to compile and link n modules without CUTTER.



Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique