MAX-PLANCK-INSTITUT FÜR PLASMAPHYSIK

GARCHING BEI MÜNCHEN

GALE Programmer's Handbook

Erich Müller
Robert Lathe
Klaus Kottmann

IPP R/24 October 1977

Die nachstehende Arbeit wurde im Rahmen des Vertrages zwischen dem Max-Planck-Institut für Plasmaphysik und der Europäischen Atomgemeinschaft über die Zusammenarbeit auf dem Gebiete der Plasmaphysik durchgeführt.

GALE Programmer's Handbook

E. Mueller
R. Lathe
K. Kottmann

Abstract

This manual is intended for the applications programmer, who wishes to use GALE supported services in user tasks. These services are implemented as additions to the programming language being used in the form of subroutine calls (FORTRAN and MACRO-11) or Macro expansions (MACRO-11).

Table of Contents

Preface

0.1 Manual Objectives

This manual is intended for the applications programmer, who wishes to use GALE supported services in user tasks. These services are implemented as additions to the programming language being used in the form of subroutine calls (FORTRAN and MACRO-11) or Macro expansions (MACRO-11).

0.2 Structure of the Manual

This manual is organized into chapters. All chapters describe data Input/Output in any form. The first two chapters are concerned with data retrieval and graphic data display, and the remaining chapters are provided to describe terminal I/O.

CHAPTER 1

Introduction

As outlined in the "Terminal User's Guide" GALE (General Acquisition system for Labratory Experiments) is understood as a functional and logical extension of the RSX-11M Operating System. Besides the capability of data acquisition and preservation and of monitoring and control, some facilities are offered, for use by the GALE application programmer, but which may also be of interest for any other programming.

The programmer using these facilities must of course be familiar with the programming language and compiler being used, and also the Task Builder. Familiarity with the GALE Terminal commands as described in the "Terminal User's Guide" is required as well as in certain instances further aspects of the GALE system, in which case the appropriate document will be referred to as needed.

Subroutines called from FORTRAN or MACRO are kept either in the system object library or in object libraries containing the appropriate package of utility subroutines. In the latter case, the user task must be built with the proper object libraries in the source specifications. Macro definitions required for assembler programming are without exception contained in the System Macro Library RSXMAC.SML, and thus may simply be defined by invoking the .MCALL assembler directive.

The GALE system provides program support in the following areas:

Retrieval of data from data files generated by the GALE acquisition process.

Graphic terminal display with a palette of options and features.

General input from the user terminal.

Printing messages and user-formatted data on the users terminal and console log device.

CHAPTER 2

Data File I/O Interface

2.1 Introduction

The I/O interface to GALE data files is comprised of several subroutines, which allow the user in a simple way to access all the logical blocks contained in the data files. All these subroutines are supplied to the user in the library DASLIB.OLB. In addition to the reader assumptions made in the introduction of this manual, the reader should be familiar with the GALE data file structure described below when using the GALE data file I/O interface. This chapter contains descriptions of the following routines:

OPFIL Open GALE data file.

CLFIL Close GALE data file.

GETDAT Get data from GALE data file.

GETHDR Get Header Block (HDR).

GETDDB Get Diagnostic Descriptor Block (DDB).

GETMCB Get Module Control Block (MCB).

GETNXT Get ID codes of the next dependent and independent CNF-block of a given CNF-block.

GETSHT Get shot number of last GALE data file.

2.2 GALE Data Files

One of the basic assumptions made in the GALE design, is that the administration effort associated with data files can be greatly reduced if all of the data acquired from one experiment is contained in one data file and if the resulting files are named in such a way that they can be conveniently identified.

2.2.1 File Naming Convention

As discussed in the GALE Terminal User's Guide, all files are designated with names of the format:

name.extension;version

This is also true of GALE data files. It should be noted that the files originally created by the ACQ task all have the version number 1. In the naming of a data file, the extension element is set to the short-form (3-character) experiment name. The name element always consists of a single letter (at present "B") followed by five digits. The letter indicates the file structure level of the file in question. The following digits give the sequential experiment number associated with the file.

NOTE

The name of a file is also held in the file header record so that the file name can be re-constructed if the data file is inadvertently renamed.

2.2.2 File Structure

It has been implied above, that all of the data derived from one experiment are stored in one data file. Data files must therefore contain adequate information to enable the Data File I/O Interface to locate the data requested by user tasks. This has been achieved by imposing the local system configuration onto the structure of the data file. A system configuration may be viewed as a binary tree structure, an

example of which is given in Figure 2-1. The various nodes in the structure represent diagnostics and their modules. In Figure 2-2, the structure of a corresponding data file is shown; it is evident that the nodes containing the data are linked to the module nodes from which the data was obtained. The actual layout of the resulting file is shown in Figure 2-3.

In a system such as GALE, where many tasks simultaneously access a given data file, the amount of I/O needed to obtain a given datum should be minimized. An expediant means for achieving this, given the structure used in GALE data files, is through the use of random access methods. GALE data files consist of nodes made up of one or more 64 byte records. (Eight such records may be read with one read access). The data file I/O interface takes advantage of this structure and therefore is able to skip large portions of the file when a piece of data is requested. For example, if a task requests data from diagnostic 2, module 3 in the file depicted in Figure 2-3, the following steps would be carried out:

1. Read the "Header" node and obtain the pointer to the "Diagnostic 1" node.

2. Read the "Diagnostic 1" node. This is not the requested diagnostic, so obtain the pointer to the next diagnostic node.

3. Read the next diagnostic node. This is the requested diagnostic, so obtain the pointer to the first associated module node.

4. Read the module node. If this is not the required node, obtain the pointer to the next module node and retry this step. If this is the "Module 3" node, then go to the next step.

5. Read the requested data and return them to the requesting task.

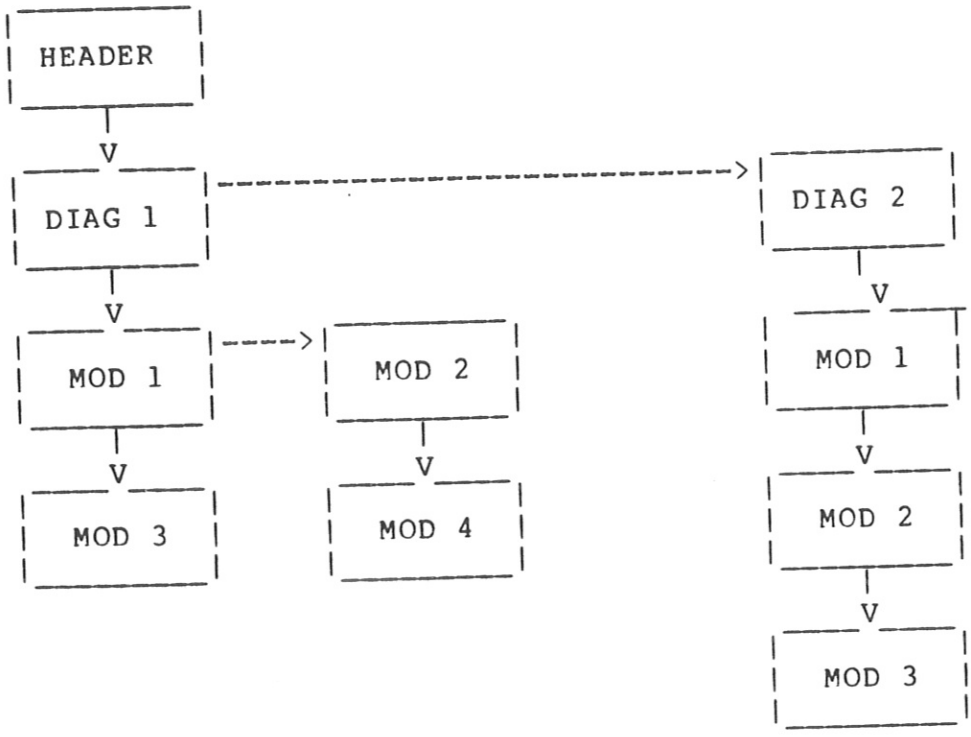


Figure 2-1

Example System Confiuration

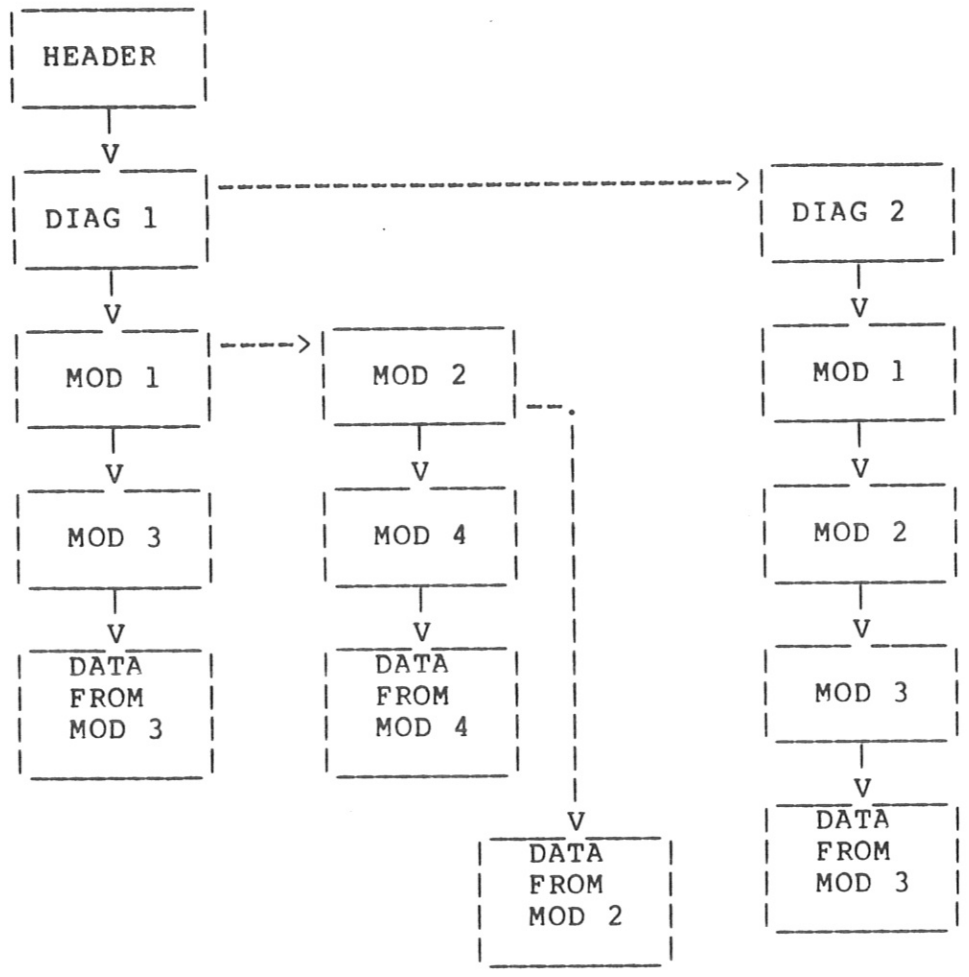


Figure 2-2

Data File Block Linkage

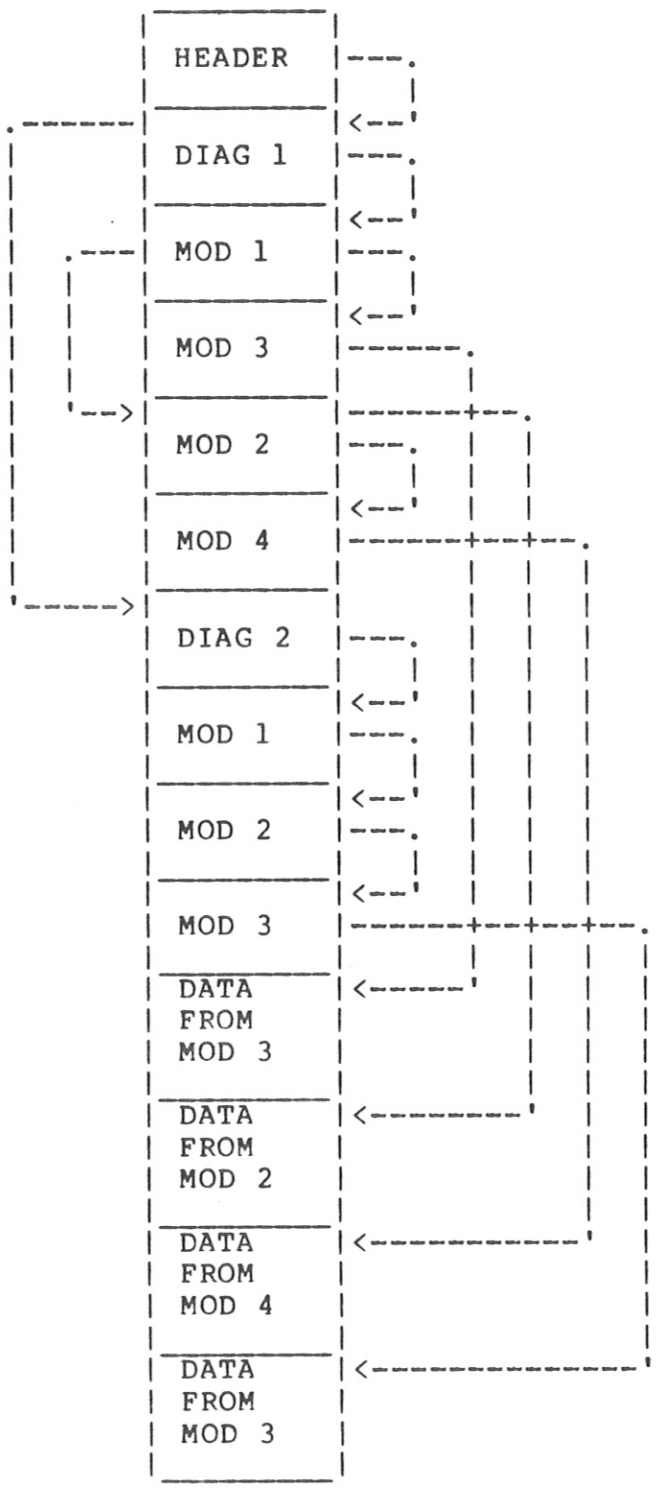


Figure 2-3

Data File Structure

2.3 General Requirements

Before any GALE data file may be accessed via the routines listed above, the user should consider the following:

1. All routines forming the I/O interface to GALE data files are callable from FORTRAN programs.

2. All interface routines are gathered in the library DASLIB.OLB. Usage of these routines therefore requires at task build time the input file specification DASLIB/LB or DASLIB/LB:routine1:routine2:... respectively.

3. The GALE data and sytem devices DD0: and DS0: must be assigned globally to physical devices, e.g.:

```
ASN DK1:=DD0:/GBL
ASN DK2:=DS0:/GBL
```

4. GALE data files are accessed by the interface routines for record I/O. It is the users responsibility to provide a sufficiently large File Storage Region (FSR). FSRs may be allocated and extended via the FSRSZ$-Macro or by issuing the EXTSCT or ACTFIL statements in the optional keyword input to the Task Builder (TKB). The procedures are described in the "RSX-11M I/O Operations Reference Manual" and the "RSX-11M Task Builder Reference Manual". Note that the GALE data file I/O interface routines allow access of one file at a time only.

5. The routines described below use some subsidiary routines, also contained in DASLIB.OLB. In order to speed up access time in overlay structures the user should have these utilities in his root segment. This is accomplished by specifying

```
DASLIB/LB:GETCOM
```

   in the root-segment specification of the Overlay Descriptor Language (ODL) file.

6. A task using the GALE data file interface routines must link to the common area DASCOM.OBJ, which is accomplished via the task builder (TKB) input:

```
DS:[SYSUIC]DASCOM.OBJ
```

7. The interface routines use Logical Unit Number (LUN) 2 in accessing GALE data files. Therefore LUN 2 is not free for other usage until CLFIL has been called.

8. To use the interface routines there must of course be at least one GALE data file resident on the specified data storage medium.

The first call in accessing a GALE data file has to be the call for routine OPFIL. The last one may be the call for CLFIL, although it is not mandatory (open files are closed automatically on task exit). The order in which all other routines are called is not restricted.

2.4 Open GALE Data File (OPFIL)

This routine assigns LUN 2 to the specified data device and opens a GALE data file. Before accessing a data file with any other interface routines OPFIL must be called:

```
CALL    OPFIL(SHT,IOSB[,DEV])
```

SHT — Shot number of the data file to be opened.

IOSB — I/O status block. This is a vector of 2 Integer*2, which is set by OPFIL. Upon return, a value of +1 indicates successful completion; any other value indicates an error. Error codes and their meaning are listed further below.
The second Integer of the I/O status block is set to the number of bytes transferred to the callers buffer. Because OPFIL does not transfer any data this argument is not used.

DEV — Optional vector of 4 Logical*1 values containing the device name string of the device where the data file resides. If this argument is omitted the GALE data device DD0: is used by default.

2.5 Close GALE Data File (CLFIL)

CLFIL closes a GALE data file previously opened by OPFIL. In addition LUN 2 is freed for other usage. The calling sequence is of the form shown below, where the same argument description applies as with routine OPFIL.

```
CALL    CLFIL(SHT,IOSB[,DEV])
```

2.6 Get Data from GALE Data File (GETDAT)

The GETDAT routine is designed to access logical blocks of data delivered by the associated experiment. Data to be read are considered to be a sequentially ordered vector like a FORTRAN singly dimensioned array. Any amount of data may be requested starting at any point in the logical block. The call is as follows:

```
CALL    GETDAT(DAT,LEN,STRT,FMT,IOSB,DID,MID)
```

DAT Address of the user buffer.

LEN Length of user buffer in data items.

STRT Starting data item in the logical block (STRT>=1). This parameter is incremented each time a data item is transfered to the user buffer and must be set equal to or greater than 1, for the first of successive calls to GETDAT.

FMT Transfer control argument, where the low byte serves as the buffer and conversion descriptor and the high byte serves as the MUX data descriptor.

Bit 00 Request for byte data.
Bit 01 Request for Integer*2 data.
Bit 02 Request for 4 byte data.
Bit 03 In addition to Bit 02 above this bit indicates, whether the request is for Real*4 (bit set) or for Integer*4 (bit clear).
Bit 08 Request for MUX data.
Bit 12
. Channel number of requested MUX data
Bit 15

Bits not listed above are not used, but must be set to zero.

IOSB I/O status block. This is a vector of 2 Integer*2, which is set by GETDAT. Upon return, a value of +1 indicates successful completion; any other value indicates an error. Error codes and their meaning are listed further below.
The second Integer of the I/O status block is set to the number of bytes transferred to the callers buffer. This is true, even in case the first word indicates an error.

DID Diagnostic ID code of the data block requested.

MID Module ID code of the data block requested.

2.7 Get Header Block (GETHDR)

GETHDR enables the user to retrieve all data contained in the Header block of the GALE data file currently open. The calling sequence is as follows:

```
CALL    GETHDR([BLK],[PAR],[PLN],IOSB,DID)
```

BLK Optional user buffer address for the fixed part of the Header. The length of the buffer must be at least 40 bytes. If this argument is not specified the fixed part of the Header will not be returned.

PAR Optional user buffer address for the system parameters contained in the Header block. If this argument is not specified no system parameters will be returned.

PLN Length in bytes of the buffer PAR. PLN must be specified if PAR is specified and may be omitted if PAR is not specified.

IOSB I/O status block. This is a vector of 2 Integer*2, which is set by GETHDR. Upon return, a value of +1 indicates successful completion; any other value indicates an error. Error codes and their meaning are listed further below.
The second Integer of the I/O status block is set to the number of bytes transferred to the callers system parameter buffer PAR. This is true, even in case the first word indicates an error.

2.8 Get Diagnostic Descriptor Block (GETDDB)

GETDDB returns to the callers buffer all data contained in a Diagnostic Descriptor Block given by its ID number. The call is as shown below:

```
CALL    GETDDB([BLK],[PAR],[PLN],IOSB,DID)
```

BLK — Optional user buffer address for the fixed part of the DDB. The size of the buffer must be at least 48 bytes. If this argument is not specified the fixed part of the DDB will not be returned.

PAR — Optional user buffer address for the user parameter portion of the DDB. If this argument is not specified user parameters are not returned.

PLN — Length of the user buffer PAR in bytes. PLN must be specified if PAR is specified and may be omitted if PAR is not specified.

IOSB — I/O status block. This is a vector of 2 Integer*2, which is set by GETDDB. Upon return, a value of +1 indicates successful completion; any other value indicates an error. Error codes and their meaning are listed further below.
The second Integer of the I/O status block is set to the number of bytes transferred to the callers buffers. This is true, even in case the first word indicates an error.

DID — Diagnostic ID code of the DDB requested.

2.9 Get Module Control Block (GETMCB)

GETMCB returns to the callers buffer all data contained in a MCB given by its Diagnostic and Module ID number. The call is as shown below:

```
CALL    GETMCB([BLK],[PAR],[PLN],IOSB,DID,MID)
```

BLK — Optional user buffer address for the fixed part of the MCB. The size of the buffer must be at least 40 bytes. If this argument is not specified the fixed part of the MCB will not be returned.

PAR Optional user buffer address for the device dependent parameter portion of the MCB. If this argument is not specified user parameters are not returned.

PLN Length of the user buffer PAR in bytes. PLN must be specified if PAR is specified and may be omitted if PAR is not specified.

IOSB I/O status block. This is a vector of 2 Integer*2, which is set by GETMCB. Upon return, a value of +1 indicates successful completion; any other value indicates an error. Error codes and their meaning are listed further below.
The second Integer of the I/O status block is set to the number of bytes transferred to the callers buffers. This is true, even in case the first word indicates an error.

DID Diagnostic ID code of the MCB requested.

MID Module ID code of the MCB requested.

2.10 Get Next Blocks (GETNXT)

GETNXT is useful in getting information about the diagnostic configuration of the experiment as it was at data taking time. This routine returns at each call the ID codes of the next dependent and independent CNF-block of a given CNF-block, specified by its Diagnostic and Module ID codes. The call is of the form:

```
CALL    GETNXT([DID],[MID],DEP,INDP,IOSB)
```

DID Optional Diagnostic ID code. If this argument is not specified the links of the Header are returned.

MID Optional Module ID code. If this argument is not specified the links of DDB given by DID are returned.

DEP ID code of the dependent block. If DID was not specified, DEP contains upon return the ID code of the first DDB in the structure. If MID was not specified, DEP is set to the ID code of the first MCB in the Diagnostic given by DID. If DID and MID were given the ID code of the MCB which is linked through B.LK2 to the current MCB is returned in DEP.

INDP ID code of the independent block. If DID was not specified INDP is set to zero, because the Header block has no independent link at present. If MID was not specified, INDP is set to the ID code of the next DDB linked to the present DDB. If DID and MID were specified, INDP holds upon return the ID code of the MCB, which is linked through B.LK1 to the current MCB.

IOSB I/O status block. This is a vector of 2 Integer*2, which is set by GETNXT. Upon return, a value of +1 indicates successful completion; any other value indicates an error. Error codes and their meaning are listed further below.
The second Integer of the I/O status block is not used.

2.11 Get Shot Number (GETSHT)

GETSHT returns to the caller the shot number of the last allocated GALE data file. This is done by reading the value at offset H.LST in the Header block of the GALE configuration file (CNF). The call is of the form:

```
CALL    GETSHT(IOSB)
```

IOSB I/O status block. This is an vector of 2 Integer*2, which is set by GETSHT. A value of +1 indicates successful completion; any other value indicates an error condition. Possible error codes and their meaning are listed below. The second Integer*2 is set to the requested shot number.

NOTE

GETSHT does not access any data file and may be called only if no data file is currently open. This is true through the fact, that GETSHT uses LUN 2 in accessing the GALE configuration file.

2.12 Error Return Codes

The routines forming the I/O interface to GALE data files use the following error codes, which differ from standard RSX-11M interpretation. All other error codes returned have standard RSX-11M meaning.

NOTE

The error codes are returned as Integer*2 values.

Module	Error Code	Meaning
OPFIL	IE.BNM	Invalid shot number
CLFIL	IE.BAD	Either the shot number or the device name string do not reflect the parameters of the open GALE data file.
GETDAT	IE.BAD	Invalid starting data item or transfer control argument specified, or Real conversion error.
	IE.SPC	Illegal user buffer or data buffer address not specified.
	IE.EOF	End of logical data block.
	IE.BYT	Byte request for data which do not fit into a byte.
	IE.NSF	Logical data block not found.
	IE.WAT	Format argument erroneously specified.
	IE.BTP	Unprocessable length of data item in associated MCB.
	IE.NFI	Bad DID or MID specified.
	IE.BCC	MUX data sequence error, e.g. datum of channel n not followed by datum of channel n+1.
GETHDR GETDDB GETMCB	IE.EOF	End of block detected.
	IE.NFI	Bad DID or MID specified.

2.13 Programming Hints

Due to the very generalized structure of the GALE data files and the associated file handling routines, there are a large number of ways for locating the data desired - and also a number of possibilities where the programmer can make errors. The following points give some suggestions which have proven useful to beginning programmers.

2.13.1 Using GETSHT

The GETSHT routine, in addition to providing the sequential number of the latest data file, also provides a convienient means for synchronizing data taking and data processing when used in conjunction with the OPFILE routine. An experimentalist may thus start a processing program utilizing this facility and then automatically process and view the results of an experiment immediately after the data has been taken. Such a program might incorporate statements like the following:

```
C
C       FORTRAN PROGRAM SECTION FOR SYNCHRONIZING
C       DATA ACQUISITION AND DATA PROCESSING
C
        DIMENSION       ISTB(2)
        NXTSHT=0
        LSTSHT=0
        GOTO    101
C
C       WAIT 5 SECONDS IF THERE IS NO NEW DATA
C       OR IF THE CURRENT DATA ACQUISITION PHASE
C       IS NOT YET COMPLETED.
C
100     CALL    MARK (10,5,2)
        CALL    WAITFR (10)
C
C       GET THE NEW FILE NUMBER
C
101     CALL    GETSHT (ISTB)
C
C       IF THERE IS AN ERROR, THEN STOP
C
        IF (ISTB(1) .NE. 1) STOP
C
C       DETERMINE THAT THIS IS REALLY A NEW FILE.
C       IF NOT, WAIT A WHILE AND TRY LATER.
C
        NXTSHT=ISTB(2)
```

```
        IF (NXTSHT .EQ. LSTSHT) GOTO 100
C
C       TRY TO OPEN THE FILE.
C
        CALL    OPFILE (NXTSHT,ISTB)
C
C       IF SUCCESSFUL, GO PROCESS THE DATA.
C
        IF (ISTB(1) .EQ. 1) GOTO 102
C
C       THE ACQUIRE TASK MAY STILL BE WRITING DATA INTO
C       THE FILE (SHOWN BY ERROR CODE = -29).  IF SO,
C       RELEASE ACCESS AND TRY AGAIN LATER.  IF NOT, THEN
C       THERE HAS BEEN A FILE SYSTEM ERROR, SO STOP HERE.
C
        IF (ISTB(1) .NE. -29) STOP
        CALL    CLFILE (NXTSHT)
        GOTO 100
C
C       SET THE LAST FILE NUMBER TO THE CURRENT FILE NUMBER.
C
102     LSTSHT=NXTSHT
C
C       READ AND PROCESS DATA HERE !!!
C
C       CLOSE THE FILE
C
        CALL    CLFILE (NXTSHT)
C
C       AND TRY TO GET NEW DATA.
C
        GOTO 101
```

2.13.2 Specifying the DID and MID

A number of the routines described in this chapter require the specification of the Diagnostic and Module Identification Codes (DID and MID, respectively). The methods available for determining the proper DID and MID range from elementary to very sophisticated.

In the simplest case, the DID and MID may be fixed into the data processing program. This is often the best technique for special purpose data processing where the processing is always applied to data from a specific diagnostic and module. After the experiment configuration has been specified, any priveleged user may obtain the DID and MID values desired by simply listing the diagnostic using the DLG task.

More sophisticated programmers may take advantage of the fact that GALE data files are structured as linked binary lists. A program might be written which asks the terminal user to input the diagnostic name and module designation whose data are to be processed. If the diagnostic is called LASER and the module QD2 delivers the data, then the combined use of the routines GETNXT, GETDDB and GETMCB in a binary tree search would provide a means for obtaining the desired DID and MID. The following steps would be required:

1. Use GETNXT with the DID and MID unspecified to obtain the DID of the first diagnostic in the system.

2. Use GETDDB specifying the DID previously obtained to read the contents of the DDB.

3. Determine if this is the diagnostic LASER by checking the contents of the diagnostic name entry. If not, go to step 4, otherwise the diagnostic has been found so go to step 5.

4. Use GETNXT specifying the DID obtained previously to obtain the DID of the next diagnostic, then go to step 2.

5. Use GETMCB specifying the previously obtained DID and MID to read the current MCB.

6. Determine if this is the module QD2 by checking the contents of the MCB device and unit entries. If this is the desired MCB, then data may now be read since the DID and MID are identified. If not, go to step 7.

7. Use GETNXT with the current DID and MID to get the next MID, then go to step 5.

CHAPTER 3

Graphic Display

3.1 Introduction

A graphic display facility is provided which enables the user to plot data on a terminal. Plot data are produced by a run-time package, which is designed for interactive or frame by frame usage, and are output by the Plot Task (PLT). Frame by frame usage requires each plot sequence to be enclosed in a CALL FRAME and CALL ENDFRA respectively, whereas interactive usage requires only one call. The following routines of the run-time package are available:

FRAME	Define scale of the page.
ENDFRA	Ends plot data production.
PLOTL	Draw a curve connecting points with a line.
PLOTLS	Draw a curve connecting points with a dashed line.
PLOTP	Display predefined characters at given coordinates.
PLOTSY	Plot text at a given coordinate.
SCALE	Define new scale.
CRT011	Do interactive plot.
CRT021	Round a given range for scaling.
CRT031	Scale and subdivide a given range.
ICON	Convert Integer to ASCII representation.
NEWPAG	Erase screen.
URSPR	Home cursor.

3.2 General Requirements

Before producing any plots with the graphic display facility the user should consider the following:

1. The run-time package produces an intermediate data set (a disc file) containing the plot data which are interpreted and output by the Interpreter Task (PLT). The UFD associated with the plot system, UIC[1,7], must have Read, Write, Extend and Delete priveleges for all users. This may be accomplished by establishing the UFD with a command such as

   ```
   UFD DK0:/PRO=[RWED,RWED,RWED,RWED]
   ```

2. To function properly, PLT must be task built as shown below:

   ```
   TKB>SY:[1,7]PLT/PR:0=OD:[1,101]INTHPT
   TKB>OD:[1,101]INTER/LB,IORLIB/LB
   TKB>/
   ENTER OPTIONS:
   TKB>ASG=DK0:1
   TKB>ASG=TI:2:4:5
   TKB>ASG=CO:6
   TKB>TASK=...PLT
   TKB>//
   ```

   where SY: denotes the RSX-11M system disk and OD: denotes the disk where the objects, supplied by PDE, reside.

3. After PLT is built as outlined above it must be installed and started:

   ```
   INS [1,7]PLT<CR>
   PLT<CR>
   ```

4. A user task producing plot output via the graphic display facility must link to the respective run-time routines. All these routines are contained in the library RUN.OLB which is supplied by PDE. Furthermore Logical Unit Number 5 is used by the run-time routines for terminal output, hence the user task must assign LUN 5 to the terminal (TI:).

5. The intermediate data set produced by the run-time routines and interpreted by PLT is bit-compatible with the Micro Fiche intermediate plot data set handled at the IPP Computer Center (IBM360/91 and AMDAHL460/V6). This fact allows the user to produce the data set at the computer center and, after transmitting the data set to the PDP11, to interpret and output the plot data at the local installation.

3.3 Run-Time Package

The run-time package computes all the data needed for the required plot and stores them into a disk file. Disk files are named PLT.DAT;ver, where ver is the version number which is the sum of 100 plus the physical unit number of the terminal from which the requesting task was started. When the plot data file is ready to be processed by the interpreter, the run-time package sends a message to PLT and suspends the requesting task, which is resumed by PLT when all plot data are output. The following gives a description of how to use the various routines forming the run-time package. Arguments enclosed in brackets ([and]) are optional and may be omitted. The defaults, which apply if such arguments are not specified, are given in the description of each routine.

3.3.1 FRAME - Define a Plot Page

This routine defines the scale of the page and erases the screen before a plot. A CALL FRAME must be the first call in a sequence of plotting routines, when using the run-time package on a frame by frame basis. The call is of the following format:

```
CALL FRAME(XMIN,YMIN,XMAX,YMAX[,CTRL[,REP]])
```

XMIN Minimum value for X-coordinates (Real*4).

YMIN Minimum value for Y-coordinates (Real*4).

XMAX Maximum value for X-coordinates (Real*4).

YMAX Maximum value for Y-coordinates (Real*4).

CTRL Page format control parameter (Integer*2), where the following values are acceptable:

0	Broad page format (default value).
1	Upright page format.
2	Quadratic page format.
10	For use in interactive mode only (no screen erase, but possible operator answers are displayed on top of the screen).

REP Repetition factor (Integer*2) specifying the number of copies of the plot. (The default value is zero).

3.3.2 ENDFRA - Finish a Plot Page

This routine ends the production of plot data, closes the plot data file on disc, sends a message to the Interpreter Task (PLT), thus requesting plot output, and suspends its associated task. A CALL ENDFRA must be the last call in the sequence of plotting routines. The call takes the form shown below, where no parameters are required.

```
CALL ENDFRA
```

3.3.3 PLOTL - Plot a Continuous Line

This routine draws a curve and connects the points with a line (linear interpolation). The call is of the form:

```
CALL PLOTL(X,Y,N[,INT])
```

X X-coordinates of the curve to be drawn (Real*4).

Y Y-coordinates of the curve to be drawn (Real*4).

N Number of coordinate pairs (Integer*2).

INT Integer*2 value defining the intensity of the plot (0<=INT<=28). The default value for INT is 15 and should be used for screen plotting.

3.3.4 PLOTLS - Plot a Dashed Line

This routine draws a curve and connects points with a dashed line (linear interpolation). The call is of the form:

```
CALL PLOTLS(X,Y,N[,INT])
```

X — X-coordinates of the curve to be drawn (Real*4).

Y — Y-coordinates of the curve to be drawn (Real*4).

N — Number of coordinate pairs (Integer*2).

INT — Integer*2 value defining the intensity of the plot (0<=INT<=28). The default value for INT is 15 and should be used for screen plotting.

3.3.5 PLOTP - Point Plot a Line

PLOTP displays a predefined character at the given coordinates without connecting them. The call has the format:

```
CALL PLOTP(X,Y,N[,CHAR[,SIZE[,ALPHA[,INT]]]])
```

X — X-coordinates of the curve to be drawn (Real*4).

Y — Y-coordinates of the curve to be drawn (Real*4).

N — Number of coordinate pairs (Integer*2).

CHAR — Integer*2 value containing the ASCII character in the high byte. This character is automatically centered at the given coordinates. The default character is a dot ".".

SIZE — Size of the character in mm (Real*4). Any size from zero up to 180 mm is possible, where the default value is 1 mm.

ALPHA — Inclination of the specified character in degrees (Real*4). Any inclination from zero to 360 degrees is allowed, where the default value is zero.

INT — Integer*2 value defining the intensity of the plot (0<=INT<=28). The default value for INT is 15 and should be used for screen plotting.

3.3.6 PLOTSY - Plot Text

This routine plots a text string starting at a given coordinate. The call is of the form:

```
CALL PLOTSY(TXT,X,Y[,SIZE[,ALPHA[,BETA[,R[,INT]]]]])
```

TXT ASCII string with max. 256 characters. The last character of the text has to be a $-sign. During plot execution the $-sign will be melted away.

X X-coordinate (REAL*4) of the lower left point of the first character of the text.

Y Y-coordinate (REAL*4) of the lower left point of the first character of the text.

SIZE Size of the characters in mm (Real*4). Any size from zero up to 180 mm is possible, where the default value is 1 mm.

ALPHA Inclination of the specified text line in degrees (Real*4). Any inclination from zero to 360 degrees is allowed, where the default value is zero.

BETA Inclination of each character within the text line (cursive) in degrees (Real*4). BETA defaults to zero, where legal values range from zero to 30 degrees.

R Ratio of the width to the heigth of the characters in the text line (Real*4). This value must be in the range of 0.5 <= R <= 1, where the default value is 1.

INT Integer*2 value defining the intensity of the plot (0<=INT<=28). The default value for INT is 15 and should be used for screen plotting.

3.3.7 SCALE - Define Scale Factors

This subroutine may be used to define a new scale. The call is of the following form, where the parameters have the same meanings as described in FRAME above.

```
CALL SCALE(XMIN,YMIN,XMAX,YMAX)
```

3.3.8 NEWPAG - Erase the Terminal

This routines performes a page erase for terminal plotting. The call is of the following form, where no parameters are required.

```
CALL NEWPAG
```

3.3.9 URSPR - Set Cursor Home

This routine is of interest only for terminal plotting. It places the cursor in the upper left corner (home position). The call is of the form:

```
CALL URSPR
```

3.3.10 ICON - Convert Integer to ASCII

This routine converts a given Integer to a signed ASCII string, which is a decimal representation of the integer. Each converted digit string consists of five characters which are sign, three digits and a trailing $-sign. For example the binary value -123 will be converted by ICON to the ASCII string "-123$".

```
CALL ICON(INT,TXT)
```

INT Integer*2 to be converted.

TXT Pointer to buffer where the converted ASCII string is to be stored.

3.3.11 CRT011 - Interactive Plot System

Interactive usage of the plot system may be invoked through a call to routine CRT011. A curve will then be drawn in the specified mode. Only those points are displayed which are inside the window specified. Also a title and the X- and Y-scale will be written, and a box will be drawn around the curve. The box has 750 steps in the X-direction and 500 steps in the Y-direction. For example, this will be a rectangle of 124.5 * 95 mm on a TEK4006.

The plot is terminated with a prompt for one of the following responses:

A	for automatic Y-axis scaling
S	for new scaling
C	for hardcopy
E	for exit (return to the calling program)

The format of the call is as follows:

```
      CALL CTR011(CODE,TLEN,TITLE,XTLEN,XTIT,YTLEN,YTIT,
     *            XMIN,XMAX,YMIN,YMAX,VLEN,XVECT,YVECT)
```

CODE Integer*2 value defining the plotmode, where

Bit 00	= 0	Erase screen before plot
	= 1	Overwrite old plot
Bit 01	= 0	Plot with linear X-axis
	= 1	Plot with logarithmic X-axis
Bit 02	= 0	Connect points with a line
	= 1	Display only given points

TLEN Length of the title in bytes including $-sign (Integer*2).

TITLE ASCII string displayed as title of the plot.

XTLEN Length of the X-axis title in bytes including $-sign (Integer*2).

XTIT ASCII string representing the X-axis title.

YTLEN Length of the Y-axis title in bytes including $-sign (Integer*2).

YTIT ASCII string representing the Y-axis title.

XMIN Minimum value for X-coordinates (Real*4).

YMIN Minimum value for Y-coordinates (Real*4).

XMAX Maximum value for X-coordinates (Real*4).

YMAX Maximum value for Y-coordinates (Real*4).

VLEN Number of coordinate pairs (Integer*2).

XVECT Real*4 array containing the X-coordinates.

YVECT Real*4 array containing the Y-coordinates.

3.3.12 CRT021 Auto-scale an Axis

This routine replaces the input parameters with rounded values suitable for scaling of diagrams. The difference of the input parameters is rounded, so that it can be expressed with 1, 2.5 or 5 times a power of ten. The upper and lower limits are the rounded to zero or a value which is a multiple of 1/5 of the computed difference and returned to the caller.

```
CALL CRT021(AMIN,AMAX)
```

AMIN Minimum value (Real*4).

AMAX Maximum value (Real*4).

3.3.13 CRT031 Scale an Axis

This routine computes a scale factor, so that a given minimum and maximum value can be expressed as a value between -999 and +999, i.e. as a value with three significant digits. From that range 6 equally spaced TIC mark values are computed. The call is of the form:

```
CALL CRT031(AMIN,AMAX,ITIC,IEXP)
```

AMIN Minimum value (Real*4).

AMAX Maximum value (Real*4).

ITIC Vector of 6 Integers*2 which is filled with equally spaced TIC mark values.

IEXP Power of ten by which the TIC mark values are considered to be multiplied to conform to AMIN and AMAX.

CHAPTER 4

Command Line Input

4.1 Introduction

The Command Line Input facilities accomplish all the logical functions required to enter 80 byte command lines dynamically during programm execution for the MACRO-11 programmer. Input is accepted from the users terminal as well as from indirect command files. Command lines may be analyzed and parsed, if input is expected in standard RSX command syntax. These facilities serve as a standarized interface for obtaining and interpreting dynamic Command Line Input.

4.2 Get Command Line (GCML)

The Get Command Line routine (GCML) embodies all the logical capabilities required to enter 80-byte command lines dynamically during program execution. Before command lines may be input using the GCML routine, the user must consider the following:

1. The pseudo device MO: must be known to the system and must be mounted.

2. LUN 5 must be assigned to the users terminal TI: and LUN 1 must be assigned to the pseudo device MO:. Assignments may be done dynamically using the ALUN$ Macro, or at task build time via the ASN option.

3. The user must link the object module of GCML to his task. This is accomplished by specifying IOLIB/LB as an input file at task build time.

4. As outlined in the RSX11M manual "I/O OPERATIONS", at any given time there must be an FSR block buffer available for each file currently open for record I/O operations. The block buffer requirements of GCML must be considered when issuing the FSRSZ$ macro for additional record I/O.

4.2.1 Input Requirements

GCML may be called directly from MACRO-11 routines only. To request command line input the user must issue the following statement:

```
CALL    GCML
```

Registers R1 and R2 are used as parameter registers. They are provided to enable the user to have a prompt string by which GCML indicates its readiness to accept input, where

R1 gives the length in bytes of a user defined prompt string, and
R2 holds the pointer to the user defined prompt string.

If the user does not wish to have a special prompt, both registers should be cleared, which causes GCML to use

```
GCM>
```

as default prompt string.

4.2.2 Output from GCML

After GCML has been called as outlined above, it tries to get command lines from either the user's terminal or an indirect command file. A command line found is returned to the user with

R1 set to the length of the string read in
R2 set to point to the string read in

where the carry bit in the Processor Status Register is cleared.

For example, if the user executes the following code

```
          .
          .
10$:      CLR     R1              ;INDICATE DEFAULT
          CLR     R2              ;PROMPT STRING
          CALL    GCML            ;GET DATA FILE NAME
          BCS     EXIT            ;END INPUT, SO EXIT
          CALL    PROC            ;PROCESS DATA FILE
          BR      10$             ;LOOP
            .
            .
```

GCML prompts to solicitate the users input:

```
GCM>B00321.PDE<CR>
```

After return R1 gives the length of the string input and R2 holds the pointer to it. On receiving a "^Z" input GCML returns with carry set, to indicate end of input to the calling program, which then may take the appropriate actions. In the example above, the data file names may also be stored in a disk file. In this case, instead of typing in a new data file name each time GCML is called, the user specifies the file containing the data file names as an indirect command file:

```
GCM>@DK1:FILNAM.PDE<CR>
```

GCML in response will return the first record contained in the file FILNAM.PDE residing on DK1:. Successive calls cause GCML to read successive records of the file until one of the following occurs:

1. The end-of-file (EOF) is detected on the current indirect file. In this case the current indirect file is closed, the command input level count is decremented by 1, and the previous command file is re-opened. If the command input level count is already 0 when EOF is detected, GCML returns with the carry bit set.

2. An indirect file specifier is encountered in a command line. In this case, the current indirect command file is closed, and the new indirect file is opened. The first command line therein is then read and returned to the user.

4.2.3 Error Conditions

An error condition detected during execution of command line input is reported to the user via the message output (MO) and causes GCML to request another command line. As stated above, a carry set condition upon return indicates end of input or end of file respectively, rather than an error condition.

4.3 Command String Interpreter (CSI)

The CSI routine analyzes command lines, which may be input by the Command Line Input routine (GCML) and parses them into their component data set descriptors which are required by the File Control System (FCS) for accessing files. The user, who wishes to analyze and parse command lines via the CSI routine must link the object module to his task by specifying IOLIB/LB as an input file at task build time.

4.3.1 Input Requirements

CSI processes command lines in the following formats only:

1. dev:[g,m]output filename.type;version/switch

   More than one such file specification can be specified by seperating them with commas.

2. dev:[g,m]output filename.type;version/switch,...=
   dev:[g,m]input filename.type;version/switch,...

Registers R1, R2, R3 and R4 are used as parameter registers and must be preset as described below.

R1 length of command line
R2 address of command line
R3 indicates whether an input file specifier (R3=0) or an output file specifier (R3<0) is to be parsed next
R4 address of the associated switch descriptor table. If no switches are to be processed R4 must be cleared. The format of the switch descriptor table is described in the RSX11M manual "I/O OPERATIONS".

4.3.2 Output from CSI

After CSI has been called, it removes non-significant characters from the input string and checks it for syntactic validity. During execution R1 and R2 will be overwritten to contain the following values:

R1 address of the dataset descriptor of the current file specifier.

R2 flag bits indicating the status of the current CSI operation. The following expressions are defined globally and may be used to determine the status of operation.

CS.EQU indicates that an equal sign has been detected during the check of the command line, signifying that both, output and input file specifiers where present.

CS.NMF indicates that the current file specifier contains a file name string.

CS.DIF indicates that the current file specifier contains a directory string.

CS.DVF indicates that the current file specifier contains a device name string.

CS.WLD indicates that the current file specifier contains an asterisk(*), signalling the presence of wild card specifications.

CS.MOR indicates that the current file specifier is terminated by a comma and that more file specifiers are to follow.

CS.ERR indicates that an irrecoverable syntax error has been detected during validation of the command line.

4.3.3 Error Conditions

An error condition detected during execution of command string interpretation is indicated by the carry bit in the Programm Status Register (PS). Carry clear indicates successful completion, where carry set means that one of the following error conditions has occured:

1. A switch was given with a file specifier, but the address of the switch descriptor table has not been supplied by R4, or the switch descriptor table does not contain a corresponding entry for the switch.

2. An invalid switch value has been specified.

3. More values accompany a given switch in the file specifier than there are corresponding entries in the switch descriptor table for decoding those values.

4. A negative switch is present in the file specifier, but the corresponding entry in the switch descriptor table does not allow the switch to be negated.

CHAPTER 5

Namelist Input/Output

5.1 Introduction

A facility is provided which allows user modifications of data at run-time in a most comfortable fashion, where users are not concerned with the various types of variables, i.e. Integers, Floating-point, Logical or Strings. For more detailed information see IPP Report R/19 "Beschreibung einer Namelist Routine unter RSX-11M".

5.2 General Requirements

Before using the Namelist facility (NML) the user should consider the following:

1. NML uses the Message Output Task described in chapter 6. Therefore the pseudo device MO: must be known to the system and must be mounted.

2. If the Help function (see below) is required the user is responsible for mounting MO: with the proper User Message File.

3. Namelist routines also use the Command Line Input (CLI) facility described in chapter 4. As outlined there, CLI does record I/O, so that in the event that the user intends to do other record I/O, the File Storage Region must be defined accordingly.

4. In order to use NML, a task must link to the proper object modules, which are contained in the libraries NML.OLB and IOLIB.OLB. Furthermore, Logical Unit Number (LUN) 1 must be assigned to the pseudo device MO: and LUN's 5 and 6 must reflect the standard assignment to the task's terminal TI:.

5.3 Namelist Input/Output

Activation of Namelist I/O is achieved via a call to the main routine of the NML facility, which takes the following format:

```
CALL NMLIST(NCB,IOF,ABO)
```

NCB — Pointer to any one of the NCBs forming a list associated with the data to be modified by this call (Integer*2).

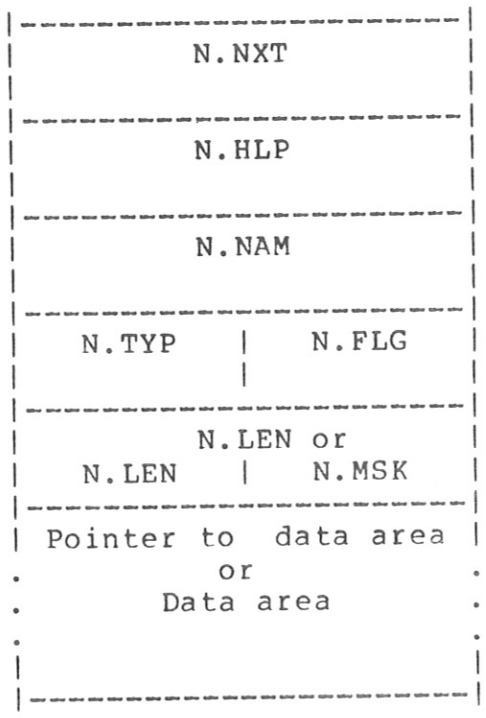
IOF — Integer*2 variable which is set to ASCII "I" if input and output are desired, or set to ASCII "O" if only output is wanted.

ABO — This parameter contains upon return a value of -1 if a "^Z" input was detected. In all other cases it is cleared. ABO is an Integer*2.

5.4 Namelist Control Block (NCB)

In order to enable NML to dynamically modify data, each variable must be described by a Namelist Control Block (NCB). All NCBs associated with variables which are to be modified by a single call to routine NMLIST are linked into a circular list as shown below:

Each NCB contains the name of its associated variable, format and length of the data associated with the variable, some control information and a pointer to the data area or the data themselves. NCBs are constructed conforming to the following format:



N.NXT Pointer to next NCB (Integer*2).

N.HLP Starting record number of a format string in the User Message File, which holds an explanatory text for the variable associated with this NCB (Integer*2).

N.NAM Name by which the data of the associated variable are referred to by NML (4 bytes RAD50 notation).

N.FLG Logical*1 control variable, which is set to NS.RMT (=1) if the NCB contains a pointer to the data (remote) or set to NS.LCL (=0) if data are located within the NCB (local). Additionally NS.HLP (=2) must be set if N.HLP contains a meaningful record number. For example if data are remote and a Help function is available N.FLG is set to 3 (NS.RMT!NS.HLP).

N.TYP ASCII data format code (Logical*1), where the codes A (ASCII string), D (decimal Integer*2), E (Real*4), I (decimal Integer*1), O (octal Integer*2), Y (octal Integer*1) and B (bit variable) are defined.

N.LEN Associated variable data region length in bytes. This is an Integer*2 value, except for N.TYP set to "B". In this case N.LEN is a Logical*1 set to 1 or 2 respectively depending upon whether the bit is to be modified in a byte- or word- variable.

N.MSK Logical*1 bit indicator, which is required only if N.TYP is set to "B". N.MSK gives the position of the bit to be turned on or off (0<=N.MSK<=15).

N.OFS Pointer to the associated data region if NS.RMT is set or start of the data area if NS.LCL is set.

5.5 NCB Macro

An NCB as described above may be simply allocated and initialized via the NCB macro. All symbolic offsets and expressions concerned with an NCB are defined in the macro BLKDF$, which is expanded before operating on any NCB. The NCB macro call is of the form:

```
label:  NCB     nxt,hlp,nam,flg,typ,ofs,len,msk
```

label Label which defines the entry to the NCB and which is used by another NCB as a linkage pointer.

nxt Pointer to next NCB (see N.NXT).

hlp Help function starting record number (see N.HLP). If not specified a comma must be given as a place holder.

nam Variable name used by NML (up to 6 ASCII characters).

flg Control flag (see N.FLG).

typ Data type format code (see N.TYP).

ofs Pointer to data area if data are remote. Otherwise not specified (comma must be given as place holder).

len Length of data area in bytes (see N.LEN).

msk Bit indicator for B-type data (see N.MSK).

In the example below an NCB is shown which refers to an Integer*2 variable named HEIGHT (not labeled!) which immediately follows the NCB:

```
NCB1:   NCB     NCB2,,<HEIGHT>,NS.LCL,D,,2
        .WORD   0
NCB2:   NCB     ........
```

If the data area of the variable is remote and is an Integer*2 vector with 10 elements the macro call is as follows:

```
NCB1:   NCB     NCB2,,<HEIGHT>,NS.RMT,D,H,20.
                .
                .
H:      .BLKW   10
```

5.6 Error Conditions

The Namelist routines report errors via the Message Output facility rather than returning an error code. Error messages have the format:

```
NML -- text
```

Possible errors are listed in the GALE Terminal Users's Guide.

CHAPTER 6

Message Output ACP

6.1 Introduction

A facility is provided for outputting formatted messages. Its goals are:

1. To provide a system wide standard for message output with emphasis on error reporting.

2. To keep the code per task as small as possible and provide coherent information.

Besides the construction of messages, the following considerations should be noted:

1. The MO task (task name = MO....), has two message destinations; the console log device (CL:) and the terminal associated with the requesting task (TI:).

2. Output from the MO task to the destinations is asynchronous; requests to the MO task may be synchronized via a WAIT Directive.

6.2 User Task Interface to MO Task

Message output is initiated via a QIO Directive to the device MO:. The MO task is considered to be a device handler, and as such it is a privileged task, with access to system data bases.

As with all handlers, a requesting task must assign a LUN to device MO: in order to use it. The LUN can be assigned in two ways.

1. The user task can execute the ALUN Directive to device MO:.

2. The user can assign the LUN via the task builder command, as:

   ASG=MO:n

   where n is specified as the LUN value.

Several macros have been supplied in the system macro file (RSXMAC.SML) to simplify the user task QIO initiation to the MO task. These macros are descibed below.

6.2.1 Format String Descriptor

The format string, from which a message is constructed using MO, is described by a pointer to it and the length of that format string, if included in the user program (Fig 6-1 and 6-2). However, if the format is in a disk file, the format string descriptor is not significant (Fig 6-3).

6.2.2 Paramter List

The message handler has the capability of inserting user arguments in a predefined message. A pointer to the parameter list is therefore provided in the macro calls and is the address of a table of sequential arguments to be used in constructing the message. The format of the table depends on the code used in the message format string and is described below.

6.3 MO Task Operation

After the MO task has dequeued the request node set up by the user QIO Directive, message processing proceeds as follows.

1. The format string is moved from the user task or specified file to the MO task area.

2. The message string is created in portions of 64 bytes (see message construction below).

3. The output portion is queued to one or both of the destination devices.

4. The request node is released declaring I/O done, when the last portion is output on the requestor terminal, or the console log device, if this was the only destination.

NOTE

If output is to be sent to both destinations, I/O done is declared after completing the request for the user task's terminal. Output to the console log device on behalf of the user task, for this case, is always asynchronous to that task's execution.

6.4 Message Construction

Messages are constructed from formatted strings that can be stored either in the user task space or in a disk file, with a fixed record length of 64 bytes. The format string consists of fixed and variable characters. In order to construct a message, the user must supply values to be substituted for the variables. An example for a format string is:

```
ALPHA:%8A
```

The format string is scanned and each character is copied into a message buffer until an escape (%) character is encountered. This triggers an interpretation of the next few characters according to the following syntax:

```
%count code

    or

%V code
```

where:

count — is a numeric ASCII string that is converted into a positive decimal integer indicating how many times the action indicated by the code character is to be performed. If not specified, 1 is assumed.

V — is used to indicate that the count is variable and the next word in the parameter list is interpreted as the count.

code — is a single letter indicating the action to be performed. The code values are summarized in Table 6-1.

In the example, %8A is interpreted to mean:

Move 8 characters from the buffer whose address is the next item in the parameter list.

In the example %VA, the interpretation is:

Get the count, a variable number, from the current position of the parameter list pointer. Increment the pointer and move the variable number of bytes or characters as stated above.

In the example below LN1 is the format string address and PAR1 is the address of the parameter list. To pass format string information to the MO task, the user task proceeds as follows:

1. Assume the formatted message was assembled as:

```
LN1:    .ASCIZ  /ALPHA:%8A/      ;FORMAT STRING
```

2. During program execution a parmeter list contains the pointer to the ASCII string.

```
PAR1:   .WORD   ASTR             ;PARAMETER LIST
                 .
                 .
                 .
ASTR:   .ASCII  /ABCDEFGH/
```

3. The user task issues a macro call:

```
        MOUT$S  #LUN,#FLG,#IOSB,#LN1,#100,#PAR1
```

4. MO responds to the macro call and outputs the following message to the requesting task's terminal:

```
ALPHA:ABCDEFGH
```

Figures 6-1 and 6-2 show more complex format strings, the macro calls to format them, and the resulting output string. The following sections give a description of the macros used to initiate message output and describe default values that are used to construct the DPB when parameters are omitted.

Table 6-1

Code	Meaning
%nA	Move n ASCII bytes from the buffer whose pointer is taken from the current location of the parameter list.
%nS	Insert n blanks (spaces) at the current position in the output buffer.
%nD	Convert n words beginning at the current location in the parameter list. Each word produces a signed zero-suppressed ASCII string (maximum sign and five digits) that is a decimal representation of the word.
%nE	Convert 2*n words beginning at the current location in the parameter list. Each word-pair produces a signed zero-surpressed ASCII string (in FORTRAN format 1PE12.4) that is a floating point representation of the word-pair.
%nI	Convert n bytes beginning at the current location in the parameter list. Each byte produces a signed zero-suppressed ASCII string (maximum sign and three digits) that is a decimal representation of the byte.
%nO	Convert n words beginning at the current location in the parameter list. Each word produces a signed, zero-suppressed ASCII string (maximum six digits) that is an octal representation of the word.

%nY Convert n bytes beginning at the current location in the parameter list. Each byte produces a signed, zero-suppressed ASCII string (maximum three digits) that is an octal representation of the byte.

Each converted digit string consists of seven characters including leading blanks. Omitting the sign, implies +.

%nP Erase screen. There is always only one page erase, even if n is greater than one.

%nN Insert n line terminators (CR,LF) in the output string.

%nR Convert n words in RADIX-50 notation to ASCII representation beginning at the current location in the parameter list. Each word produces an ASCII string of three characters.

%nT Get the current time parameters and convert them into ASCII where hours, minutes and seconds are represented in decimal (hh:mm:ss). Time is always shown only once, even if n is greater than one.

NOTE

After byte conversions triggered by format codes I or Y, the parameter list is always accessed on an even address for the next conversion by MO. For example, if a format string specifies:

...%3I,%3Y,...

both, the first byte for decimal and also the first byte for octal conversion is taken from a word boundary.

```
        MOUT$S  #LUN,#FLG,#IOSB,#LN1,#100,#PAR1
                                .
                                .
                                .
;
;  FORMAT STRING
;
LN1:            .ASCIZ  /ALPHA:%6A,%N,DEC:,%2D,%2N,OCT:%3O/
                                .
                                .
                                .
;
;  PARAMETER LIST
;
PAR1:           .WORD   ASTR            ;STRING POINTER
                .WORD   123.            ;ARGUMENTS FOR
                .WORD   456.            ;DECIMAL CONVERSION
                .WORD   111             ;ARGUMENTS FOR
                .WORD   222             ;OCTAL CONVERSION
                .WORD   333
                                .
                                .
                                .
ASTR:           .ASCII  /ABCDEF/
```

The resulting message is as follows:

```
ALPHA:ABCDEF
DEC:     123     456
OCT:     111     222     333
```

Figure 6-1
Example Using Counts in the Format String

```
        MOUT$S  #LUN,#FLG,#IOSB,#LN2,#100,#PAR2,,#1
                                .
                                .
;
;  FORMAT STRING
;
LN2:        .ASCIZ  /ALPHA:%VA,%N,OCT:,%VY,%N,DEC:%VD/
                                .
                                .
;
;  PARAMETER LIST
;
PAR2:       .WORD   6                   ;STRING LENGTH
            .WORD   ASTR                ;STRING POINTER
            .WORD   3                   ;NUMBER OF OCTALS
            .BYTE   111                 ;ARGUMENTS FOR
            .BYTE   177                 ;OCTAL CONVERSION
            .BYTE   -1
            .EVEN
            .WORD   2                   ;NUMBER OF INTEGERS
            .WORD   123.                ;ARGUMENTS FOR
            .WORD   456.                ;DECIMAL CONVERSION
                                .
                                .
ASTR:       .ASCII  /ABCDEF/
```

The resulting message is as follows:

```
ALPHA:ABCDEF
OCT:    111     177     377
DEC:    123     456
```

Because the destination parameter was set greater zero the message is also shown on the system log device, where time and task name preceed the message:

```
13:24:51 SEN2> ALPHA:ABCDEF
OCT:    111     177     377
DEC:    123     456
```

SEN is the first part of the task name not equal to "..." and the appendix 2 means that the user task is associated with terminal 2.

Figure 6-2
Example Using V in the Format String and
Output to both Destinations

6.4.1 Message Files

Format strings (e.g. error messages) can be stored in a file. In this case the format string descriptor is irrelevant and the record parameter is used to obtain the format string for message construction. Formats from message files are not limited in length, but must start at the beginning of a record and must be terminated by a binary zero. Records are taken from two different files given at MOUNT time via the switch

MOU:/SMSG=file descriptor1,/UMSG=file descriptor2

where file descriptor1 denotes the System Message File and file descriptor2 denotes a file containing user defined messages. (See also message macro description, GALE Terminal Users's Guide and GALE System Programmers Handbook).

In the examples below, R1 contains the address of the parameter list. The absolute value of the next parameter gives the starting record number of the format string within the file, where a positive number selects the System Message File and a negative number selects the User Message File.

```
                .
                .
MOUT$S  #LUN,#FLG,#IOSB,,,R1,#2 ;SYSMSG
                .
                .
                .
MOUT$S #LUN,#FLG,#IOSB,,,R1,#-17        ;USER MSG
                .
                .
```

Figure 6-3
Examples of Formats from Disk Files

6.4.2 Programming Hints

As outlined above the parameter list contains values and pointers to be used in formatting the message string. A non-privileged task must have its parameter list and also strings pointed to by it entirely in its task address space, whereas a privileged task may additionally address the first 28K of physically existent memory and the I/O-page. The MO task accepts addresses supplied by the parameter list pointer or within the parameter list as follows:

1. Each address is considered to be virtual

2. If a given address does not lie within any of the task's address windows, it is only valid if that task is privileged.

3. Addresses not within the privileged task address windows are considered to be physical addresses.

4. Addresses beyond 160000(8) supplied by privileged tasks are considered to address the I/O-page rather than the physical locations beyond 28K.

Format strings are address-checked by the system QIO-processor. The device data base of MO triggers the system to allow format strings to be byte aligned.

6.5 Message Macro Description

This section contains an explanation for each of the macros that are supplied for the users of MO. The macros are listed below.

MOUT$S
MOUT$
MOOP$S
MOCL$S

6.5.1 MOUT$S

This macro generates a DPB for initiating output with the MO task, pushes it onto the stack and executes it.

Macro call:

```
MOUT$S  lun,flg,iosb,fmt,flen,prm,rec,dst
```

Argument	Meaning
lun	Logical unit number used for pseudo device MO:.
flg	Event flag number; may be used to synchronize message output with the issuing task's execution.
iosb	Pointer to the issuing task's I/O status block, which is set on QIO completion to indicate success or failure.
fmt	Pointer to the format string; not relevant if the format string is taken from a message file.
flen	Length of format string; legal format strings range from 1 byte up to 128 bytes. The length given must be greater than or equal to the actual format string length. If the format string is ended with a binary zero, i.e., generated with a .ASCIZ directive, flen may be greater than the actual string length; otherwise, flen must be equal to the actual format string length. This parameter is not relevant if the format string is taken from a message file.
prm	Pointer to the parameter list. The parameter list is a sequential list of arguments that are used in constructing the message format string.
rec	This parameter, if set non-zero, causes the MO task to get the format string from a disk file starting with the record given by the value of "rec" rec > 0 get the format string starting at the rec'th record of the System Message File rec < 0 get the format string starting at the rec'th record of the User Message File
dst	The value of dst determines where output is to be directed: dst = 0 output to issuing task's terminal dst < 0 output to system log device dst > 0 output to both destination devices

Arguments not specified are assumed to be zero. Commas must be given as place holders, however they may be omitted right of the last specified argument.

6.5.2 MOUT$

This macro causes a DPB to be generated for initiating output with the MO task.

Macro call:

```
MOUT$   lun,flg,iosb,fmt,flen,prm,rec,dst
```

The same argument description applies as given with macro MOUT$S. The example given below constructs output by interpreting the the format string starting at the 7th record of the System Message File.

```
MODPB:  MOUT$   2,,IOSB,,,,7            ;7'TH RECORD TO TI:
                      .
                      .
                      .
        DIR$    MODPB
```

6.5.3 MOOP$S

This macro performs an OPEN for the issuing task on pseudo device MO:. It must be called before producing any output with the MO task.

Macro call:

```
MOOP$S  lun,flg,iosb
```

6.5.4 MOCL$S

MOCL$S performs a CLOSE for the issuing task on pseudo device MO:. Because a CLOSE is performed automatically for each open file at task termination time this macro call is not mandatory.

Macro call:

```
MOCL$S  lun,flg,iosb
```

6.6 Message DPB Format

The DPB format for message output is the same as for the QIO$ macro. The following rules apply:

```
WORD 2: I/O FUNCTION            ;fixed to IO.WVB
                                ;for MOUT$ and MOUT$S
                                ;fixed to IO.ACW for MOOP$S
                                ;fixed to IO.DAC for MOCL$S

WORD 4: EVENT FLAG, PRIORITY    ;priority fixed to zero

WORD 6: AST                     ;fixed to no AST entry
```

6.7 Error Conditions

The MO task is designed to inform the user of error conditions that may arise. Error conditions are accommodated as follows:

1. If destination is not specified, TI: is assumed.

2. If a format string descriptor (format pointer and format length) is given with a record parameter not equal zero, the format string descriptor is ignored.

3. If a format string is to be taken from a file and there is not enough dynamic memory to read one file block (512 bytes) MO notifies the user by printing the following string on the destinations:

   *** MO: INSUFFICIENT DYNAMIC STORAGE nnn

   where nnn gives the record number which has been given at the time the error occurs. Note, that the return code in the I/O status block, if present, is set to indicate success.

4. Other errors detected before the processing of a format string, e.g. during reading of a format string of a file, cause an error code to be set in the user status block, if present.

5. Errors detected during the processing of a format string cause the part of the message already constructed to be output immediately. An error code is set in the user status block, if present. Possible errors are:

   1. Invalid format directive or illegal format length.

   2. Illegal record number.

   3. Illegal access to parameter list.

6. An error condition is indicated by a status return code less than zero. If such a return code is negated and then is used as the record argument in a MOUT$ or MOUT$S macro call, it produces a message explaining the error that has occured.

6.8 MO Status Return Codes

The following I/O status returns are made by MO:

IS.SUC	Successful MO request
IE.BAD	Invalid format directive encountered, or illegal format string length (e.g. flen > 128), or illegal record number, or any FCS error in accessing a message file
IE.BYT	Byte address for word parameter
IE.ADP	Argument out of users address space, nonexistent memory or I/O page address. Note that a privileged task may address all physically existent memory up to 28K and even the I/O-page.
IE.ALN	OPEN already performed
IE.NLN	No OPEN performed
IE.PRI	MO: not mounted

In addition, directive errors are reported via setting the Directive Status Word $DSW with the standard RSX-11M System directive error return codes.

APPENDIX A

Recommended Logical Unit Assignments

The routines described in this manual may require up to four Logical Units. It is recommend that the applications programmer assign Logical Units as given below.

LUN 1	MO:	This assignment is mandatory for tasks which utilize the Data File interface, and is recommended for other tasks in order to avoid possible conflicts.
LUN 2	DD:	This assignment is recommended for tasks which use the Data File interface; it must be at least reserved in such cases. The OPFILE routine does a dynamic LUN assignment to the device specified, if any, or defaults to DD:.
LUN 3	SY:	This assignment is required only for user tasks utilizing the Plot system.
LUN 5	TI:	This is the default Task Builder assignment. It is required for the proper functioning of all routines involving terminal operations.

READER'S COMMENTS

NOTE: THIS FORM IS FOR DOCUMENT COMMENTS ONLY.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable and well organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please turn over

Please indicate the type of user/reader that you most nearly represent.

[ ] Assembly language programmer

[ ] Higher-level language programmer

[ ] Occasional programmer (experienced)

[ ] User with little programming experience

[ ] Student programmer

[ ] Non-programmer

If you desire to have your name put on the PDE documentation mailing list, please indicate so here.....

[ ]

NAME_____DATE_____

ORGANIZATION_____

STREET_____

CITY_____

STATE_____ZIP CODE_____

COUNTRY_____

RETURN TO:

PDE PROJEKT DATENERFASSUNG
INSTITUTE FOR PLASMAPHYSICS
D-8046 GARCHING
WEST GERMANY