MAX-PLANCK-INSTITUT FÜR PLASMAPHYSIK

GARCHING BEI MÜNCHEN

GALE System Programmer's Handbook

Robert Lathe
Erich Müller

IPP R/27 February 1978

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.

Abstract

This report provides the information needed to implement the GALE system at a user installation. The model-driven structure of the system is described with examples and the steps involved in generating the model including the necessary macros are discussed.

Table of Contents

Preface

0.1 Manual Objectives

The "GALE System Programmer's Handbook" provides all the information needed to implement the GALE system. Although the manual is primarily self-contained, the reader should be familiar with the manual "DIOS I/O Operations" and, of course, with the RSX-11M system for a full understanding of this manual.

0.2 Structure of the Manual

This manual is organized into chapters with the following contents:

Chapter 1	provides an introduction to the GALE system.
Chapter 2	describes the main control structure (CNF) of the GALE system. An example structure impleme-tation is provided in Appendix B.
Chapter 3	provides information on generating Message Files for use with the Message Output utility.
Chapter 4	describes the GALE Data File structure.
Chapter 5	discusses the GALE sysgen procedure.

CHAPTER 1

Introduction

This chapter presents a general introduction to the facilities of the General Acquisition system for Laboratory Experiments (GALE). It is intended for systems programmers implementing the system on a particular experiment and for those contemplating using GALE or designing new systems.

1.1 Design Goal

The major purpose of GALE is to serve as a translator between the experimentalist and the experimental apparatus. The translation process is accomplished by means of a combination of hardware and software modules which convert electrical signals into messages and graphs meaningful to the experimentalist on the one hand, and which converts messages and logical decisions from the experimentalist into electrical signals acceptable to the apparatus on the other hand. The translation process is, in the ideal situation, transparent, however in the practical case restrictions are imposed. In particular, the user must often have an understanding of the characteristics of the hardware used to acquire the data.

The objective of the GALE development is a single system for use in a pulsed or continuous operational mode which may be generally characterized as:

1. adaptable to varic ıs experiments,
2. flexibile at a particular experiment,
3. simple for the experimentalist.

In the pulsed operational mode, experiments are performed in a definite period of time, and following the end of an experiment, the acquired data is available for processing.

Thus only the data are acquired in real-time, the processing occurs in non-real-time. This mode is most suitable for experiments with high data input rates and short duration. In the continuous mode, the acquired data is also processed in real-time, thus providing a sort of data monitoring. This mode is more suitable for experiments with relatively low data input rates and longer duration.

1.2 Design Philosophy

In discussing the GALE system philosophy, it should be kept in mind, that the development objective is a single system adaptable to a wide variety of laboratory experiment environments. The alternative is a separate system development for each application. The design philosophy implemented in GALE centers on the assumption that the characteristics of a specific application may be modelled and therefore the acquisiton system may be tailored to the application via the model. Furthermore, once a model is constructed, it must be changeable to reflect new sitiuations arising in the application. The ease with which the model may be generated and modified determines the simplicity of the system usage by the experimentalist.

The assumption that an application may be modelled implies that the system which interprets that model must be composed of a set of (semi-) independent components. In GALE, only those components of the total system are required which are specifically or implicitly given in the application model, thus minimizing the space requirements at a given installation. The components are parameterized in such a way, that the application model acts as a mask used to extract the characteristics required for the given application.

1.3 System Features

The GALE system is designed to support a wide variety of laboratory experiment applications. In the current release, GALE 3.0, many of the features described are fully supported, some are partially supported and some are at present not supported. The degree of support is indicated with the feature description.

1.3.1 System Model

Model Driven. As discussed above and in Chapter 2, the GALE system is model driven. The complete model has provisions for local and remote diagnostics. At present, only local diagnostics are supported.

Dynamic Model Modification. The system model may be dynamically modified at any time other than during the data acquisition phase. In particular, the definition of the modules may be changed in the event of a hardware failure. This feature is fully supported.

Pulsed and Continuous Modes. The present GALE release supports pulsed mode only.

Multiple Experiment Support. In some cases, it is more economical to support several experiments from one large computer (resource sharing). For such requirements, the GALE design provides for multiple configurations where each experiment is described by a separate model. MES is at present not supported.

1.3.2 Device Drivers

Central CAMAC Processing. All requests to the CAMAC system from the device drivers are processed by central CAMAC processing routines in DIOS. At the present time, two CAMAC controllers are supported, without DMA.

Dynamic Device Driver Loading. In pulsed mode, the devices required for data acquisition are very often required for only a small percentage of the time. In GALE, device drivers are in main storage only during the time that the device is actually active, thus saving storage resources. This feature is fully supported.

User State Device Driver Testing. Device drivers being developed under the GALE system may be tested in user state. Errors in the code result in abortion of the driver, not in a system crash, an important consideration in multi-user systems. This is a fully supported feature.

1.3.3 System Protection

User Logon Protection. Access to the system is password protected. This is a recommended optional feature of RSX-11M.

Diagnostic Protection. Each diagnostic is protected from inadvertent (or intentional) modification by users other than the owner and system manager. Thus, each user has access to his diagnostic(s), is however unaware of other activities in the system. A fully supported feature.

Data File Protection. All data taken are written into files which may be read by all users; only the system manager may delete data files. Fully supported.

Automatic Data File Backup. Data files are automatically written to backup storage, either in background processing or during a specified period of time, for instance overnight. Files which are no longer on-line, are transferred from the backup storage by demand requests. This feature is at present not supported.

1.3.4 Terminal Facilities

Standard Terminal Input. All teminal input is either in the form of simple commands adhering to the RSX-11M conventions or in the form of Namelist specifications. Fully supported.

Central Message Facility. All GALE error and informational messages reside in two system files. Thus all messages are system wide and are provided in non-coded form. Fully supported.

Plot Facility. An optional plot facility is provided for use with TEK 4000 series teminals.

1.4 Hardware and Software Requirements

The GALE system operates under the RSX-11M operating system. The hardware configuration required is that which is necessary to operate an RSX-11M mapped system and the desired modules and interfaces for data acquisition. These prerequisites are described in detail in Chapter 5 in this report.

CHAPTER 2

Generation of GALE Configuration Files

2.1 Introduction

The GALE data acquisition system developed at the Institute for Plasma Physics is based upon a description model of the relevant data acquisition hardware. In addition to the individual hardware descriptions, the model also represents the system topology, i.e., the interrelationships between the various hardware components. These hardware components, referred to as "modules", may be dependently or independently related. Groups of modules (in which both dependent and independent relationships may exist) which form a logically related construct are termed a "diagnostic". The set of all diagnostics present on one experimental apparatus comprises the system model. The existence of such a model allows the conception and implementation of only one software system, controlled by the model, which meets the requirements of a variety of experiments.

Due to the fact that the model forms the heart of the GALE system, the success or failure of a given application depends upon the careful construction of the configuration model. The model is held in a file referred to as the "CNF File". Once this file is generated, model parameters may be changed via the DLG task; the structure of the model itself is however fixed and may only be changed through the construction of a new model and CNF file. The model generation consists of three basic steps:

1. Defining the diagnostics with their associated modules and the interrelationships of the system components.

2. Translating the logical system model into a MACRO source file using a set of macros.

3. Generating the CNF file.

These steps are described in detail in the remainder of this chapter.

2.2 Terminology

Before attempting to describe the GALE configuration structure and its sub-structures, it is necessary to first define a number of terms used throughout this chapter. It is of the utmost importance that the systems programmer concerned with the implementation of the GALE system have a through understanding of list processing concepts. The following concepts are presented here for review purposes and are not intended to be an exhaustive discussion of the topic.

A "node", often referred to as a "record" or "entity" is a defined unit of information. Each node consists of one or more bytes of storage, divided into named parts called "fields" or "elements". The address of a node is referred to as a "link" or "pointer".

Lists may be formed into structures of various types. A "linked list" is a set of nodes whose structural properties essentially involve only the linear (i.e., one dimensional) relative positions of the nodes. The end of such a list is reached when the pointer to the next node is zero. A "circular list" is a linked list which has the property that its last node is linked back to the first node rather than pointing to zero. It is then possible to access all of the list starting at any node contained in the list.

A more complicated structure involving nodes is a "tree". Knuth * formally defines a tree as follows:

A tree is "a finite set T of one or more nodes such that

1. There is one specially designated node called the "root" of the tree, root (T); and

2. The remaining nodes (excluding the root) are partitioned into m>=0 disjoint sets T(1),...,T(m), and each of these sets in turn is a tree. The trees T(1),...,T(m) are called the "subtrees" of the root."

* Knuth, Donald E.; The Art of Computer Programming, Addison-Wesley; Reading, Mass.; 1969.

This definition has the advantage of being recursive, however it is not very intuitive. A somewhat more intuitive definition might be achieved by redefining the second part of the above as follows:

2a. Eminating from the root are one or more pointers. The remaining nodes are linked in such a way that each node is pointed to by exactly one node, its "predecessor", and points to zero or more nodes, its "successors".

A concept of particular interest in the remainder of this chapter is the "binary tree". In a binary tree, each node has at most two subtrees, distinguished as the "left subtree" and the "right subtree".

Consideration of the definition and concept of a tree reveals the following properties:

1. The root is a unique node which has no predecessors.

2. Every node other than the root has exactly one predecessor.

3. Every node other than the root is connected to the root by a unique path such that the path begins at the root and ends at the node, and such that each node on the path other than the root is a successor of the previous node on the path.

2.3 Configuration Model Structure

As implied above, the GALE system is "model driven", that is, the system can only be "operated" when a model of the hardware is provided and the model determines the operation of the system. Before describing how a model is actually implemented, it is important that the user have an understanding of the general philosophy of the modelling structure used in GALE.

The GALE model organizes the associated hardware in such a way which can best be described as a binary tree structure. In particular, this structure takes into account the following facts about the combining of modules into diagnostics and diagnostics into a complete system.

1. A system is composed of one or more diagnostics.

2. Diagnostics are completely independent of one another as far as hardware function is concerned. (The physical phenomena they observe may be very closely connected, on the other hand).

3. Each diagnostic consists of at least one and possibly more modules.

4. Within a diagnostic, modules may be independent of one another or dependent upon other modules. The "independent" or "dependent" characteristic of a module is a function of its physical interconnections with other modules. Thus, if a module has a physical output to a second module, the second module is defined to be dependent upon the first. Independent modules are defined as those modules which have no inputs from other modules in the configuration model. As will be discussed below, dependent modules in the model always follow the modules upon which they depend.

For example, consider the combination of a Pulse Generator (PG) driving several pairs of Multiplexed ADC's (MX) and CAMAC Memories (CM) within a single diagnostic: each MX depends on pulses from the PG but is completely independent of the other MX's. Each MX has a CM associated with it (dependently), into which it stores its data. This structure may be intuitively represented by the following diagram:

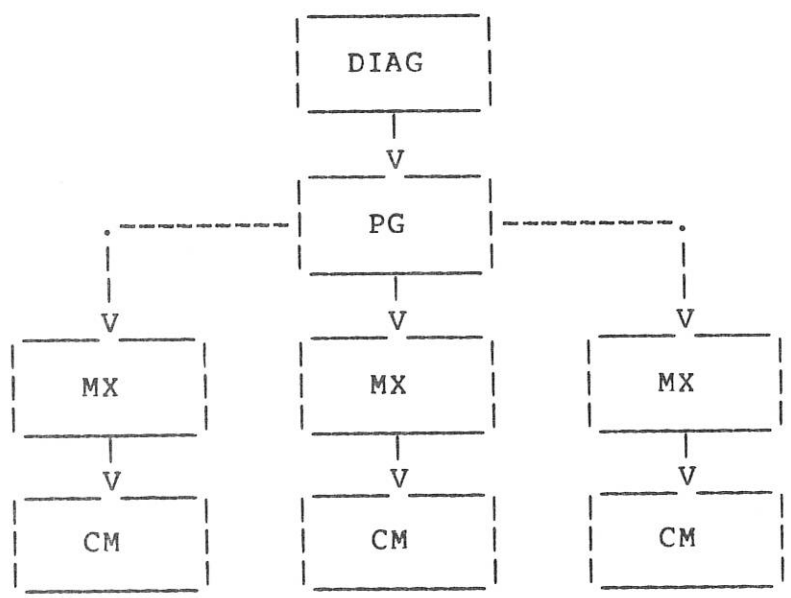


Figure 2-1
Example of a Diagnostic Structure

In the above diagram, it is evident that some of the modules are independent of others, and on the other hand, that some of the modules are dependent upon other modules. A more formal way of expressing these relationships is the concept of a binary tree. Each node in a binary tree has exactly one entry and two outputs. In the case of a GALE configuration structure, the outputs represent dependence or independence. Returning to the example above, a binary tree structure for the same diagnostic might be illustrated as follows:

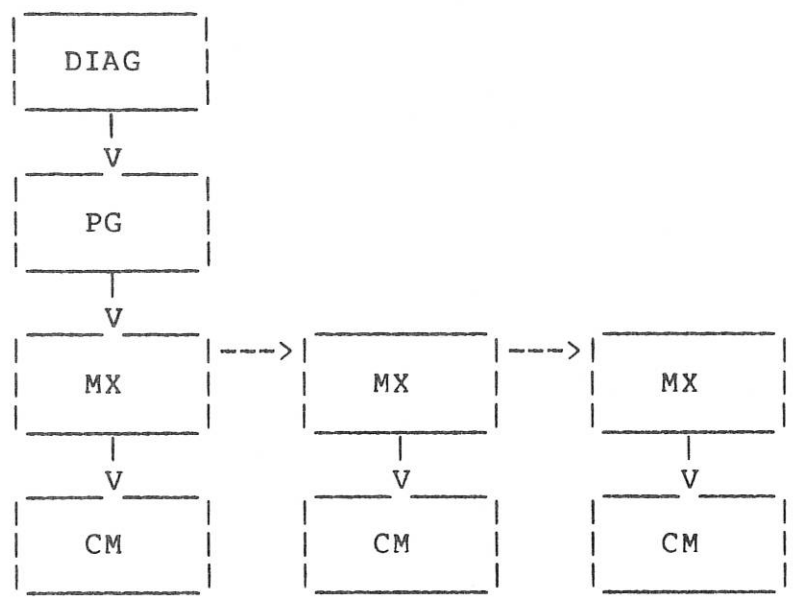


Figure 3-2
Example of a Diagnostic Binary Tree

In the above example, the downward arrows (left subtrees) symbolize the dependent relationships, the transverse arrows (right subtrees) indicate the independent relationships and the diagnostic node is the root of the tree.

A set of such diagnostics may be combined in a similiar way to form the complete configuration. Thus the GALE configuration is a binary tree of binary trees. The following figure depicts a simple system configuration.

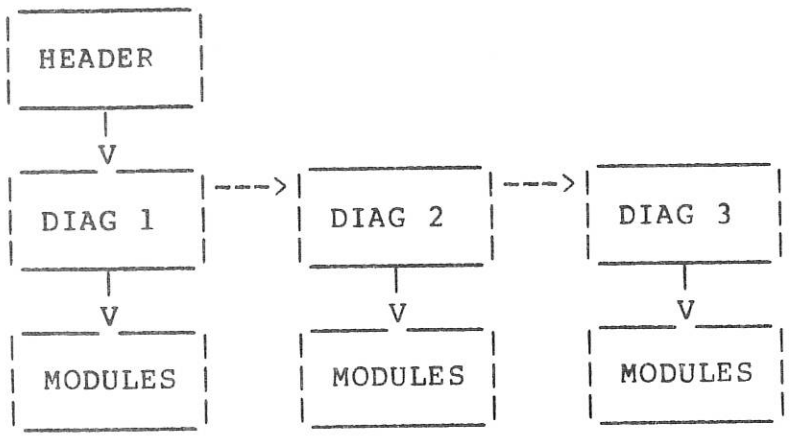


Figure 3-3
GALE Configuration as a Binary Tree

As in the previous example, the downward arrows represent the dependent relationships, the transverse arrows symbolize the independent relationships and the Header node is the root of the tree. The reader should also be aware that the Header never has an independent pointer and that the diagnostic independent pointer always is directed to the next diagnostic (if more exist).

In implementing a GALE configuration, each node of the logical configuration is converted to a control block. The following sections give detailed descriptions of the various control blocks and their contents.

2.4 Block Format

All blocks used in building the CNF file consist of multiples of 64 byte records. A set of Macros has been supplied to simplify allocation, linkage and initialization of the CNF blocks. All offsets mentioned below are listed in Appendix A. Some entries in the three different block types are concerned with the same quantities and are named by the same offsets. In particular, every CNF block begins with a common section with the following offsets.

B.LNG Block length in units of 64 bytes.

B.TYP Block type code. Identifies the block type, where

BT.HED denotes a Header Block.
BT.DDB denotes a Diagnostic Descriptor Block.
BT.MCB denotes a Module Control Block.

The block type code is set automatically when using the Header-, DDB- and MCB-Macros respectively.

B.DID In the Header block, this element is set to zero. For further information, see the descriptions D.DID and M.DID.

B.LK1 Pointer to next independent block (right pointer).

B.LK2 Pointer to first dependent block (bottom pointer).

B.NCB Pointer to associated NCBD list.

2.4.1 Header Block

The Header block contains information needed to identify the experiment and the current shot, certain system dependent information, and space for additional user information or comments not associated with a specific diagnostic.

H.FPC File processing code. Of concern for data processing purposes only. Cleared by Header Macro.

H.DAT Date of last shot in binary (day month and year). Cleared by Header Macro and initialized by ACQ.

H.TIM Time of last shot in binary (hour, minute and second). Cleared by Header Macro and initialized by ACQ.

H.LST Last shot number corresponding to H.DAT and H.TIM. Cleared by Header Macro and initialized by ACQ.

H.NXT Shot number of next data file to be written. Cleared by Header Macro and initialized by ACQ.

H.EXP 3-character ASCII experiment name. H.EXP is used to form the file name extension of the data files.

H.FLN Data File length in units of 64 byte records. Relevant in Data Files only.

H.LUP Length of system user parameter section in bytes. Automatically initialized by the Header Macro.

H.UPR Start of the system user parameters. User parameters may be specified by using the UPAR Macro.

2.4.2 Diagnostic Descriptor Block

The DDB describes a single diagnostic, which may consist of one or more modules. It contains information to identify the diagnostic to the terminal user and to the data processing programs, general characteristics, status information, user parameters and link pointers to further DDBs and to the Module Control Blocks.

D.DID Diagnostic ID code. Identifies the diagnostic by an unique number ranging from 1 to 255.

D.TYP Reserved for future use. Any type code from 1 to 255 is acceptable.

D.MLN String length of diagnostic name (1 to 22).

D.MSG Diagnostic name in ASCII (e.g. "HF HEATING").

D.UOC User member code for diagnostic.

D.UGC User group code for diagnostic. D.UOC and D.UGC form the User Identification Code (UIC) of the user owning the diagnostic. Only privileged users and the owning user have access to the diagnostic data base.

D.CHR Diagnostic characteristic bits, set to

- DC.DAQ for data acquisition diagnostic
- DC.CTL for monitoring and control diagnostic
- DC.LCL for local diagnostic directly connected to host computer.
- DC.RMT for remote diagnostic connected to subsidary computer. If neither DC.LCL nor DC.RMT are set, then the diagnostic is defaulted to local (DC.LCL).

D.STS Diagnostic status bits. Initialized to zero (diagnostic logically off-line). If set to ST.ONL the diagnostic is considered to be logically connected to the GALE system.

D.RSA Diagnostic remote station address, valid only if DC.RMT is set.

D.LUP Length of DDB user parameter section in bytes. Automatically initialized by the DDB macros.

D.UPR Start of the diagnostic user parameters. User parameters may be specified by using the UPAR Macro.

2.4.3 Module Control Block

The MCB controls the actual hardware function of the associated module during data acquisition or monitoring and control respectively. It contains information identifying the module and the diagnostic, link pointers to other MCBs within this diagnostic, module independent information used by the driver, module characteristics and module dependent parameters determining its operation mode.

M.DID Diagnostic ID code, matches code in D.DID of the associated diagnostic. Indicates the diagnostic to which the module belongs.

M.TYP 2-character ASCII module type code. This is the partial driver name for this module (e.g. MX, if the module drivers P-section is named MXDRV). DIOS uses the type code to determine the module driver.

M.UNIT Module physical unit number (0 to 255).

M.MID Module ID code in diagnostic. Identifies the module within a diagnostic by an unique number ranging from 1 to 255.

M.ACP 2-character ASCII partial task name of the ACP containing the module driver (e.g. DA if an ACP with task name DA.... holds the module driver).

M.CTL Module control bits, set to

- MC.CTL for monitoring or control modules.
- MC.GEN data are to be read from the module and stored in the data file.
- MC.PRE used in connection with MC.GEN=1 to indicate that the first request to read data from a module is to be given directly after receipt of the pre-trigger; otherwise ignored.
- MC.PST used in connection with MC.GEN=1 to indicate that the first request to read data from a module is to be given after receipt of the post-trigger; otherwise ignored.
- MC.HLD no data are to be read from the module and stored in the data file (MC.GEN=0), however, other modules which do generate data to be stored logically depend upon this module. Thus operations on the module will be terminated only after receipt of the post-trigger.

MC.INT for modules generating interrupts.
MC.CAM for CAMAC modules.
MC.SUB for sub-modules. The sub-module concept may be used by non-standard drivers for modules which are interfaced through a multiplexer module. In this case the multiplexer is considered a normal DIOS module with a CAMAC or UNIBUS address respectively in M.ADR and MC.SUB not set in its MCB. Each sub-module must be distinguished from others on the same multiplexer by specifying a physical sub-address in M.ADR. The MC.SUB bit prevents this address from being treated as an invalid CAMAC or UNIBUS address by DIOS.
MC.NPR read data in DMA mode from the module.
MC.OVF indicates that a module has generated more data than expected, which might be caused by a hardware malfunction (set by DIOS).
MC.ERR error on module (set by DIOS).

M.DLN Length in bytes of module data item.

M.DFM Format code of data delivered by the module.

MF.NST Non-standard type
MF.BYT Logical*1 data
MF.INT Integer*2 data
MF.FLT Real*4 data
MF.MUX Data in MX format
MF.NL1 Negative logic ones complement
MF.NL2 Negative logic twos complement

M.DCT Number of data items desired for MC.GEN=1; otherwise ignored. If M.DCT=0 amd MC.GEN=1, then no data will be read and the module will be treated as MC.HLD=1.

M.DAT Pointer to starting record of module data block in data file. Set by ACQ task.

M.VCT Module interrupt vector address. Legal addresses must range from 70(8) to 774(8) and must be a multiple of 4.

M.PRI Module interrupt priority for non CAMAC modules. If the module does not require interrupt service, MC.INT must not be set and M.VCT and M.PRI are irrelevant.

M.ADR Module device register or CAMAC BCNA address depending on bit MC.CAM in M.CTL.

M.ERR Module error byte. If any error has occured with the module during I/O via DIOS this byte is provided to accept the error code.

M.LUN LUN used by tasks in doing I/O to module via DIOS. DIOS assigns the LUN to the module when processing a LOAD request. Set dynamically by the task requesting I/O.

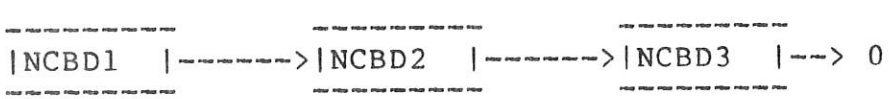
M.LPM Length of device dependent parameter section. Automatically initialized by the MCB macros.

M.DDP Start of the device dependent parameters of module. Device dependent parameters may be specified by using the UPAR Macro.

2.5 Namelist Structure

In the operation of large experiments, scientific and economical reasons necessitate the possibility of continuing work without long delays, even in the case that some acquisition or control modules become inoperable. That is, it is necessary to allow a quick exchange of modules while operation continues. This requirement is fullfilled by the CNF structure through the application of the GALE Namelist routines in conjunction with so-called GALE Namelist Control Blocks (NCBDs, which consist of slightly modified Namelist Control Blocks). With a properly constructed Namelist structure, privileged users may modify any element in the CNF structure via the DLG task.

The Namelist structure associated with the configuration model consists of a set of open-ended (non-circular) lists. These partial lists are linked according to pre-defined rules by the DLG task to form a normal (circular) Namelist. An example of such a partial list is shown below.



2.5.1 Privileged NCBDs

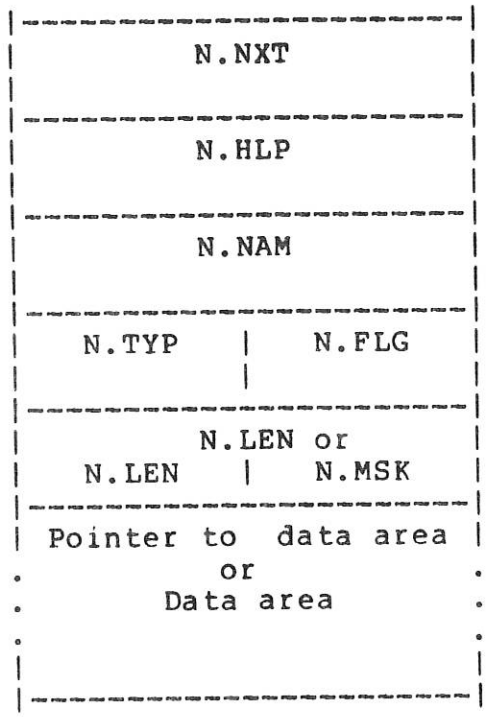
To reduce Operator errors, NCBDs provide for the modification of some entries by users with privileged access only; within one partial list these NCBDs preceed the other NCBDs. The first "privileged" NCBD holds a pointer to the first "non privileged" NCBD to allow quick access to non-privileged NCBDs. The following depicts such a partial list:

* privileged NCBD

The information contained in a part of the CNF structure consists of entries with identical meaning (e.g. module name in MCB). It is possible to modify such an entry by using a single NCBD partial list for all blocks belonging to the same type as these partial lists may be dynamically linked to the partial list associated with a specific block. Therefore, in addition to the NCBD lists for each specific block, one system-wide NCBD list is provided for each block type (Header, Diagnostic, Module). The data associated with these common NCBDs can be edited by users with privileged access rights only.

2.5.2 NCBD Block Format

NCBD blocks are used to dynamically modify their associated variables in the CNF data base. An NCBD consists of an access flag and a complete NCB as it is described in the "GALE Programmers Handbook". NCBDs are constructed conforming to the following format:



N.PRV Access privileges flag or a pointer to the first non-privileged NCBD of the partial list. If the current NCBD is not the first privileged NCBD of the partial list, this entry contains an access flag set to NS.PRV to indicate privileged access only, NS.NPR to allow non-privileged access, or NS.NDF to indicate an empty list. If the current NCBD is the first privileged NCBD of the partial list, this entry points to the first non-privileged NCBD in the partial list.

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

N.HLP Starting record number of a format string in the User Message File, which holds the explanatory text for the variable associated with this NCB (Integer*2). This is generally a symbolic name referring to a message label in the User Message file. A listing of the standard user messages is given in Appendix C.

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 always set to NS.RMT (=1). Additionally, NS.HLP (=2) is set if N.HLP contains a meaningful record number. For example if 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.

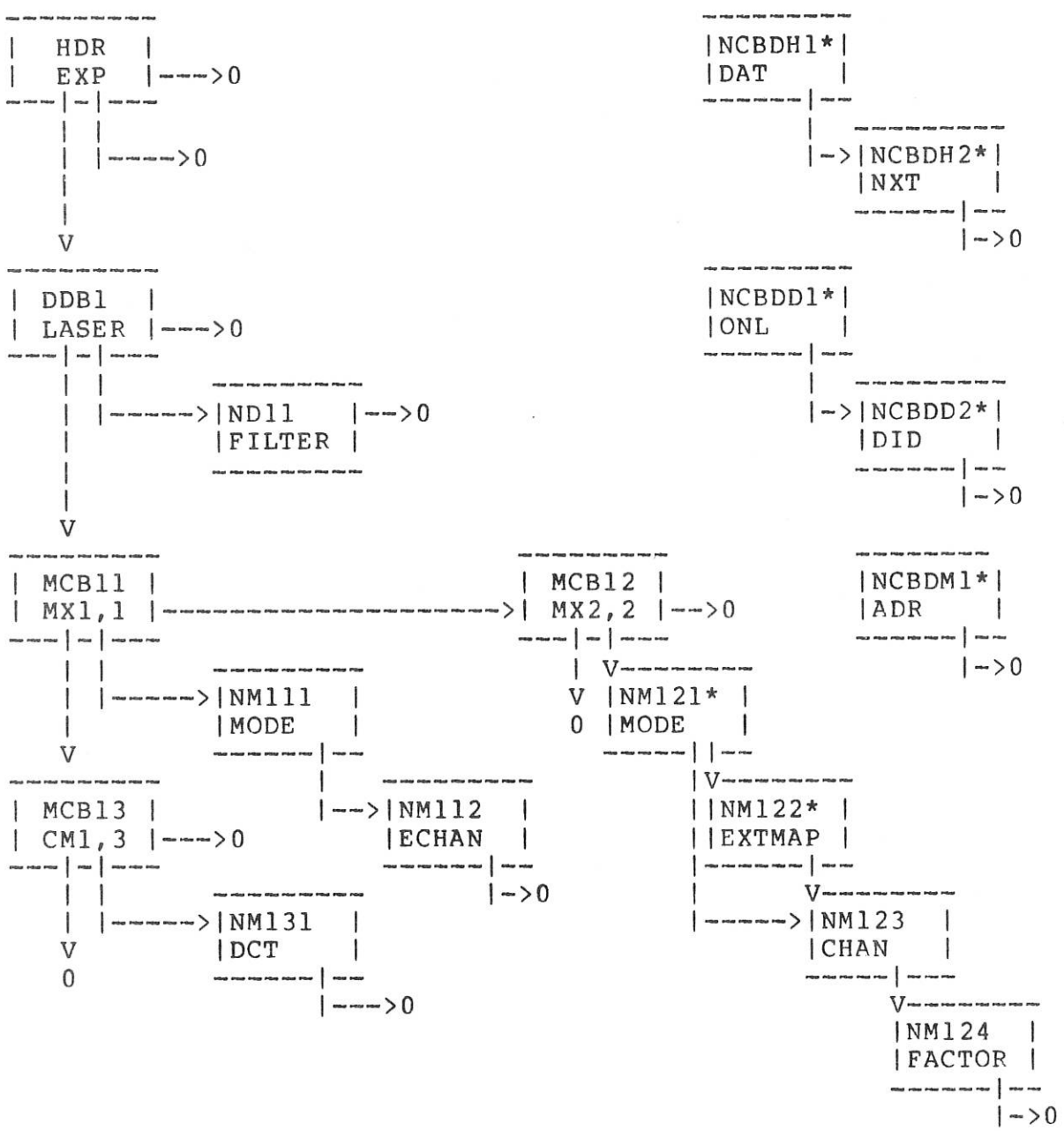
N.MSK Logical*1 bit indicator, which is required only if N.TYP is set to "B". N.MSK gives the position of the bit to be turned on or off (0<=N.MSK<=15).

N.OFS Pointer to the associated data region.

2.6 The Entire Structure

Interconnection of the CNF and Namelist structures is done via pointers. Therefore, an entry is provided in each CNF block, which holds a pointer to its associated NCBD list (B.NCB). Furthermore, the first list is fixed to belong to the Header, the second is attached to all DDBs and the third list to all MCBs of the CNF structure, thus these three system-wide lists need not be traced by pointers in the associated blocks, but are found by placement. All entries in these lists may be modified, as stated previously, by privileged users only.

For clarification an example of a structure is given below, which describes a system with one diagnostic (DDB1) consisting of three modules, where two modules (MCB11 and MCB12) are independent of oneanother and one module (MCB13) is dependent upon another (MCB11). Two entries in the Header Block (NCBDH1 and NCBDH2) and in the DDB (NCBDD1 and NCBDD2) are to be modified by users with privileged access only, also one fixed entry (NCBDM1) in all MCBs. Furthermore, the DDB should have another entry (ND11) to be modified, the first MCB two other entries (NM111 and NM112) and the third MCB one other entry (NM131), where all entries may be modified by users with normal (non-privileged) access rights. The second module should have four additional parameters (NM121, NM122, NM123 and NM124) to be modified, where two of them (NM121 and NM122) are only to be modified by users with privileged access.



* privileged NCBD

Figure 2-3
Example CNF Structure with NCBD Lists

2.7 Implementation

Implementation of the CNF structure is done in steps. First, the acquisition and control system is represented in a chart as shown above.

Beginning with the Header block, each diagnostic is represented by a DDB. The DDBs are numbered (Diagnostic ID number, DID) and given a name to yield a better survey. As seen by the CNF, all diagnostics are independent of oneanother. Next, each diagnostic is described by its structure of MCBs. In the simplest case, one diagnostic consists of one module and is therefore represented by one DDB and one MCB, where the MCB is considered to be dependent on the DDB. If a diagnostic consists of more than one module, the dependances are expressed by the linkage of the MCBs.

For example, the nth diagnostic of a system is described, where a pulse generator (PG) triggers two MUXADCs (MX), which deliver data to one CAMMEM (CM) each; the structure is represented as follows:

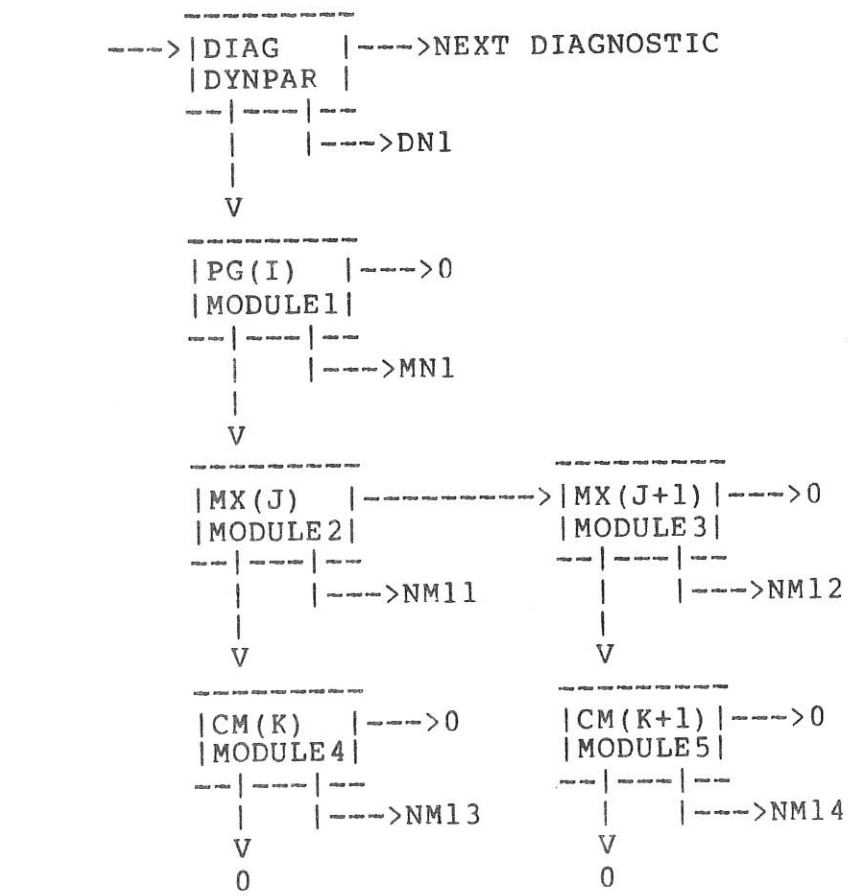


Figure 3-4
Example Diagnostic CNF Structure

The pulse generator appears as the first module, since all other modules of the diagnostic depend upon it. The next dependent module in the hierarchy is one of the MUXADCs. The other MUXADC is considered to be independent of the first one and therefore it is connected via a pointer to the right side. Again both CAMMEMs are dependent on their associated MUXADCs, therefore these modules are connected via bottom pointers. During this process, each module is provided with two numbers: the module unit number gives the number of the module of the given type within the whole system (UNIT), while the second number identifies the module within the diagnostic (Module ID number, MID). In addition, it must be determined which entries of each CNF block are to be modified via the GALE dialog task (DLG). For each such entry, an NCBD is created, where those belonging to one block are linked together, thus building a partial list. Lists may consist of NCBDs allowing normal access or of NCBDs for privileged access rights only, or of both types of NCBDs, where the privileged NCBDs preceed the other NCBDs in one list. A list built as described is provided with a global label, which is referenced by the associated CNF block.

The CNF and NCBD structures are then translated into MACRO-11 source code. Macros for this purpose are available, which are described below. In this phase of the implementation, all diagnostics and modules are described in a standarized manner; this applies also to the Namelist Control Blocks required for dynamic modifications.

Special notice is to be given to the system-wide NCBD lists which are associated with all blocks of one type. Because these lists are not linked to the CNF structure by pointers, they are allocated a fixed place within the NCBD structure, where the list for the Header starts at the first NCBD, the list for the DDBs with the second and the list for the MCBs with the third NCBD. The further arrangement of lists is free. If one of the first lists is empty then one NCBD with an access flag NS.NDF must be allocated:

```
NCBD    NS.NDF                                  ;HDR AND DDB
NCBD    NS.NDF                                  ;LIST EMPTY
NCBD    NS.PRV,,,<UNIT>,D,M.UNIT,U.BYT          ;MCB LIST
```

The entire structure is transferred into two source files ("exp"CNF.MAC and "exp"NCB.MAC; "exp" is the experiment short name), where one contains the CNF and the other the associated NCBDs. The objects produced by the assembler (MAC) are converted by the task builder (TKB, options /-HD, UNITS=0, STACK=0) into a task image file each.

In order to obtain the proper linkage between the CNF and NCBDs, it is required that each NCBD partial list be preceeded with a global label, which can be referenced by the blocks of the CNF. The image file of the CNF is therefore built using the symbol table of the image file of the NCBDs, thus inserting the proper relative addresses in the CNF. The label blocks are removed from both files so that they contain only the pure binary structure code. The resulting files are allocated for direct access with record length 16 (DASNCB."exp") or with record length 64 respectively (DASCNF."exp"), where the relative addresses of associated lists are transferred to corresponding record numbers.

Vice versa, these record numbers are converted to relative addresses again, if the files are read back into core by any GALE task guaranteeing proper linkage for the respective storage medium.

The entire structure, implemented as described above, allows all GALE tasks direct and fast access to all data. Beyond this, the NCBD structure enables the GALE Dialog task (DLG) to modify the contents of any of the control blocks of the CNF structure.

2.8 Macro Description

Macros are described below, which set up and initialize a GALE CNF model structure. They are supplied in the RSX-11M System Macro Libray RSXMAC.SML.

Arguments not specified are assumed to be zero. Leading parameters not specified, require commas as place holders. Pointers not defined or defined to zero, are interpreted as the end of linkage or they show that the corresponding block is not existent.

All macros and the symbolic values used therein are defined by the macro CNFDF$. CNFDF$ must be specified in a .MCALL directive and called afterwards (see Appendix A). All block types of the CNF structure are allocated by the following macros in multiples of 64 bytes.

2.8.1 HDR Macro

The HDR macro must be the first macro called in defining the CNF structure.

```
HDR     rcnt,ddbr,ncbr,<enam>
```

rcnt Length of Header block in units of 64 bytes.

ddbr Pointer to first DDB (B.LK2).

ncbr Pointer to associated NCBD list (B.NCB).

enam 1 - 3 character experiment code (H.EXP).

2.8.2 DDB Macro

For allocation of a DDB or MCB the user has to call two macros in sequence. The first macro defines the common part of the block structure and the user parameter section, while the second macro allocates and initializes the block specific part of the DDB or MCB respectively.

```
ddbnam: DDB     rcnt,did,ddbr,mcbr,ncbr
```

ddbnam	Name of the DDB used as reference label (e.g. DDB3).
rcnt	Length of DDB in units of 64 bytes.
did	Diagnostic ID number (D.DID).
ddbr	Pointer to the next DDB (B.LK1).
mcbr	Pointer to the first MCB (B.LK2).
ncbr	Pointer to the associated NCBD list (B.NCB).

```
        DDBE    typ,<msg>,ugc,uoc,chr,sts,rsa
```

typ	Diagnostic type code (D.TYP).
msg	Name of the diagnostic. This is an ASCII string consisting of up to 22 characters (D.MSG).
ugc	Group code of the user who is responsible for this diagnostic (D.UGC).
uoc	Owner code of the user, where UIC=[ugc,uoc] (D.UOC).
chr	Characteristic (D.CHR). This argument specifies whether the diagnostic is used for data acquisition (DC.DAQ) or is for control purposes (DC.CTL).
sts	Status, initialized to zero (off-line) (D.STS).
rsa	Address of remote station, if diagnostic is attached to a remote station (D.RSA).

2.8.3 MCB Macro

```
mcbnam: MCB     rcnt,did,mcblr,mcb2r,ncbr
```

mcbnam	Name of MCB used as reference label.
rcnt	Length of MCB in units of 64 bytes.
mcblr	Pointer to the next independent MCB (B.LK1).
mcb2r	Pointer to the next dependent MCB (B.LK2).
ncbr	Pointer to the associated NCBD list (B.NCB).

```
        MCBE    <typ>,unit,mid,<acp>,ctl,dln,dfm,dct,vct,pri,adr
```

typ	ASCII two character code for the module (M.TYP).
unit	Module unit number (M.UNIT).
mid	Module ID number (M.DID).
acp	ASCII 2 character code for the driver ACP which holds the driver for this module (M.ACP).
ctl	Module control bits (M.CTL).
dln	Length of one data item in bytes (M.DLN).
dfm	Data format key (M.DFM).
dct	Number of data items to be taken (M.DCT).
vct	Address of the interrupt vector (M.VCT).
pri	Interrupt priority (M.PRI).
adr	Module CSR address or CAMAC BCNA (M.ADR).

2.8.4 NCBD Macro

The NCBD macro allocates GALE Namelist control blocks in multiples of 16 bytes.

```
ncbnam: NCBD    acc,nxt,hlp,<vnam>,typ,offs,len,msk
```

ncbnam Name of the NCBD used as reference label.

acc Access flag set to NS.NPR for normal access and set to NS.PRV for privileged access. If a NCBD is provided for privileged access and is the first NCBD in a list where also NCBDs for normal access are present, then acc is set to the reference label of the first NCBD allowing normal access. If a NCBD list is to be defined empty, acc is set to NS.NDF.

nxt Pointer to next NCBD in the list, where for the last NCBD in a list nxt is not defined or defined to zero.

hlp Starting record number of the format string located in a message file for usage with the Message Output (MO) facility. The format string is provided to give information about the variable(s) associated with the NCBD.

vnam ASCII string which gives the variable name of which the contents are to be modified.

typ Format code, where the same codes are valid as are for the Message Output task, except codes N, P, R, S and T. Additionally the following code is specified:

B - Bit-variable

offs Offset of the variable relative to the begin of its associated block.

len Length of data area in bytes.

msk Defines for a bit variable the position of the bit to be modified in a byte (len = 1) or in a word (len = 2), where 0<=msk<=15.

2.8.5 UPAR Macro

This macro serves to specify user parameters.

```
UPAR    upn,type,value
```

upn — Offset of the user parameter relative to the beginning of its associated block. Offsets are defined by the symbols x.P01 to x.P43, where x is a placeholder for H (Header), D (DDB), or M (MCB).

type — Has the values U.ASC for ASCII strings, U.INT for 16-bit integers, U.BYT for 8-bit integers and U.FLT for reals (4 bytes).

value — Is the value which is assigned the user parameter.

CHAPTER 3

MO Message File Generation.

3.1 Introduction

The Message Output facility accepts format strings stored in two different disk files: the System Message File which holds the standard RSX-11M system error messages, and the User Message File which holds user defined messages.

For use with GALE, both files are defined and delivered to the user via the normal generation procedure. Nevertheless, the user may wish to extend message files, in particular the User Message file, for special applications. For this purpose aids are available to generate or extend message files for use with the Message Output Task.

3.2 The Source File

The source file consists of a series of macros which define the message strings and their starting points relative to the start of the file. Format strings in message files are not limited in length and may cross any number of records and even block boundaries within the file. Because format strings are located via record pointers, they have to start at the beginning of a record, and this starting point must be known globally. Furthermore, since no format string length is given explicitly, format strings in a message file must be terminated by a binary zero. The file is created in MACRO-11 source code using two macros, MSG$DF and MSG$ND which are described below.

3.3 The Message File

The Message File is a randomly accessible file with a fixed record length of 64 bytes, which is built in quite the same manner as GALE CNF files. The assembler (MAC) compiles the source code into object form and the task builder (TKB) is used to produce an image file without Task-Header, Stack and Logical Unit Table (LUT), (TKB Options: /-HD, STACK=0, UNITS=0) and a symbol table file. The message file is then a result of removing the label blocks of the image file, which may be done easily by means of the CNF task. After DisMOunting MO: and MOUnting it again with the new message file, the format strings therein are ready to be accessed. For example, if file USR.MAC holds user messages in Macro-11 source code the sequence of implementation is as follows:

```
MAC>DEV:[UIC]USR=DEV:[UIC]USR

TKB>DEV:[UIC]USR/-HD=DEV:[UIC]USR
TKB>/
ENTER OPTIONS:
TKB>UNITS=0
TKB>STACK=0
TKB>//

CNF>DEV:[UIC]USR.MSG/CF=DEV:[UIC]USR

DMO MO:

MOU MO:/UMSG=DEV:[UIC]USR.MSG
```

Tasks which desire to use messages contained in the message file should specify the global label of the message as the record number in the MO: directive. By task building the user task with the message file symbol table, the physical record number is automacally inserted by the Task Builder.

3.4 The Macros

In order to simplify message file generation, two macros are available which format the desired format strings to conform to the rules which apply to message files.

```
label:  MSG$DF  <text>
```

label represents an optional label, which may be used in locating the starting record numbers for format strings, as it is done by linking the GALE CNF structure to the associated Namelist structure (e.g. record number = label/64+1). If the label is to be used as the record number in a directive to MO:, it must be declared globally.

text denotes the desired format string or a portion of it.

Any number of MSG$DF macros may be executed in sequence to establish format strings of any length. A message file format string is properly terminated by executing the macro

```
MSG$ND
```

which has no arguments. This macro appends a binary zero to the format string established by issuing MSG$DF macros and assures that the next format string, defined by MSG$DF macros, starts at a 64 byte boundary.

CHAPTER 4

GALE Data File Structure

4.1 Introduction

The GALE Data File is allocated on the data storage device upon initiation of the ACQ task (or upon receipt of the START-TRIGGER when ACQ is in /AUTO mode) and filled with experiment data during the acquisition phase of the ACQ task. It is structured to contain all pertinent information accessible to the system about a given shot. This includes the time, date and number of the shot. Furthermore, the model of the on-line data acquisition diagnostics with the given user parameters is also written to the data file. Finally the data generated by each module are stored in the Data File, which may be understood as the Configuration File of the on-line diagnostics extended by the experimental data delivered by the data generating modules.

4.2 Block Format

The Data File is, like the Configuration File, composed of 64 byte fixed-length, randomly accessible records. The records are again grouped into blocks with the same format in the first three bytes as in the Configuration File, i.e., the length of the block in units of 64 byte records and the block type code. The types of blocks present are the following:

BT.HED	Header Block
BT.DDB	Diagnostic Descriptor Block
BT.MCB	Module Control Block
BT.DAT	Data Block

As mentioned above, the first three block types are identical to those in the Configuration File. Therefore, a summary of the essential points and differences in the various offsets will suffice here, whereas the Data Block will be described in detail.

4.2.1 Header Block

The Header Block is identical to that in the GALE Configuration File. The shot number, time and date are updated to the current shot at the time the Data File is written by ACQ. For each shot, one Data File is created, even when no diagnostics are on-line. In this case, the Data File will contain a Header Block only, where the pointer to the first DDB will be zero. As outlined in Chapter 2, the Header Block contains the shot number and the name of the experiment the Data File belongs to. Thus the Data File name may be constructed from the Header even in cases where the Data File name is destroyed. The GALE Data File name format is described in the chapter "Data File I/O Interface" in the "GALE Programmer's Handbook".

B.LK2 This pointer is changed to point to the proper record in the Data File, rather than in the Configuration File.

B.NCB The Namelist record number is meaningless.

H.DAT This entry holds the date of the shot in binary (3 bytes: day, month and year).

H.TIM Gives the time of the shot in binary (3 bytes: hour, minute and second).

H.LST Shot number of this Data File.

H.NXT Shot number of the Data File to be written after this one.

H.FLN This entry gives the total length of the Data File in units of 64 byte records.

4.2.2 Diagnostic Descriptor Block

The DDB also has the identical format as in the Configuration File. The following offsets however deserve mention:

B.LK1
B.LK2 These pointers are changed to point to the proper records in the Data File, rather than in the Configuration File.

B.NCB The Namelist record number is meaningless.

D.STS The on-line bit (ST.ONL) is always set, as only those control blocks from on-line diagnostics are copied to the Data File.

D.CHR The data acquisition bit (DC.DAQ) is always set, as only data acquisition diagnostics, as opposed to those dedicated to monitoring and control functions, are included.

4.2.3 Module Control Block

The MCB also has the same format as in the Configuration File. The following offsets deserve mention here:

B.LK1
B.LK2 These pointers are relinked to the proper MCB record numbers in the Data File.

B.NCB The Namelist record pointer is irrelevant.

M.CTL If MC.GEN is set, the module has its associated data located in a Data Block starting at the record number given in M.DAT. If MC.ERR is set, an error has occured with the module at data acquisition time, where M.ERR holds the proper error code.

M.DCT This offset contains the amount of data (in units of the format specified in M.DFM) actually read in, as opposed to the amount requested. If this number differs from the number of data originally desired, MC.ERR is set at M.CTL and M.ERR holds the proper error code.

M.DAT This offset holds the actual record number of the Data Block if the module generated any data (MC.GEN set at M.CTL). Otherwise it is set to zero.

M.ERR If MC.ERR at M.CTL is set this byte contains an error code explaining the error that has occured at acquisition time. Because this offset holds only one byte the class to which the error code belongs is not reported.

M.LUN The LUN is left over from data acquisition and is meaningless.

4.2.4 Data Block

This block type is unique to the GALE Data File. If a module delivers data to the computer (MC.GEN set at M.CTL in its MCB) the contents of the data buffer are written to a block of this format at the time specified by bits MC.PRE and MC.PST. The starting record number is maintained in the associated MCB. As for all other block types in the data file, Data Blocks are allocated in multiples of 64 byte records. The first record of each Data Block holds information about the length and type of the block, the Diagnostic and Module identification and the length of the data area in bytes.

B.LNG Length of the Data Block in units of 64 byte records.

B.TYP For a Data Block, this entry is fixed to BT.DAT.

X.DID This entry gives the ID code of the diagnostic to which the module belongs which has delivered data to the Data Block.

X.MID This entry holds the ID code of the module from which the data were read.

X.BLN Length of the following data area in bytes.

X.DAT Start of data area. Data are stored in the format as indicated by M.DFM of the module whose ID code is held in X.MID.

4.3 Block Linkage

The Header Block, DDBs and MCBs are linked in the same way as in the Configuration File.

1. B.LK1 in the Header is meaningless.
2. B.LK2 in the Header points to the first DDB.
3. B.LK1 in the DDB points to the next DDB or is zero if there are no more diagnostics.
4. B.LK2 in the DDB points to the first MCB in the diagnostic.
5. B.LK1 in the MCB points to the next independent MCB in the diagnostic.
6. B.LK2 in the MCB points to the next MCB depending upon it.

In addition, the MCBs are linked to the data blocks via M.DAT, which points to the starting record of the associated Data Block. If the module has no data, the pointer is zero. The linkage is illustrated in Figure 4-1.

4.4 Block Placement

The order in which the Header Block, MCBs and DDBs appear in the Data File is the same as it is in the Configuration File. This means that the Header Block starts at record one, followed by the first DDB with its MCBs, followed by the next DDB with its MCBs and so on. The Data Blocks follow the last MCB. They appear one after another in the order in which they were read from the modules at acquisition time, i.e., all modules giving data before the shot come before those which are read in after the shot. Figure 4-2 illustrates the placement of blocks in the Data File. Note that the pointers on the left hand side are independent and those on the right are dependent.

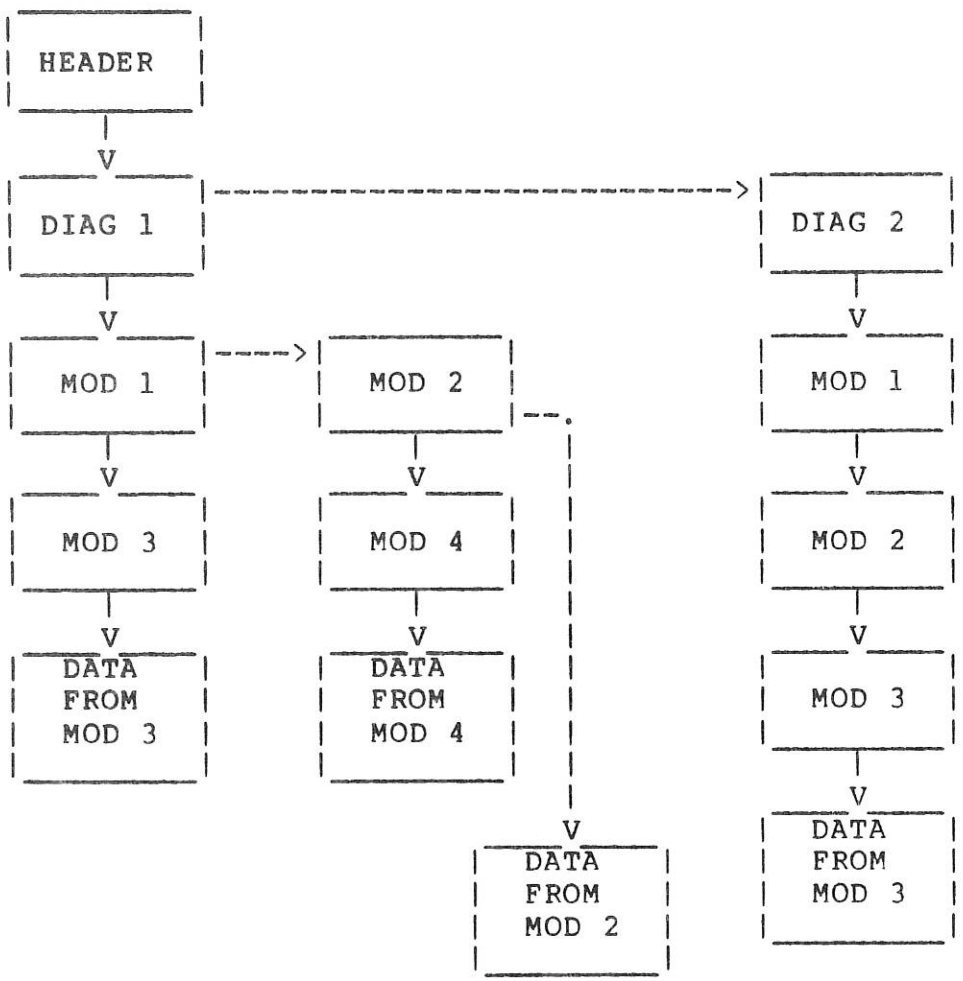


Figure 4-1
Data File Block Linkage

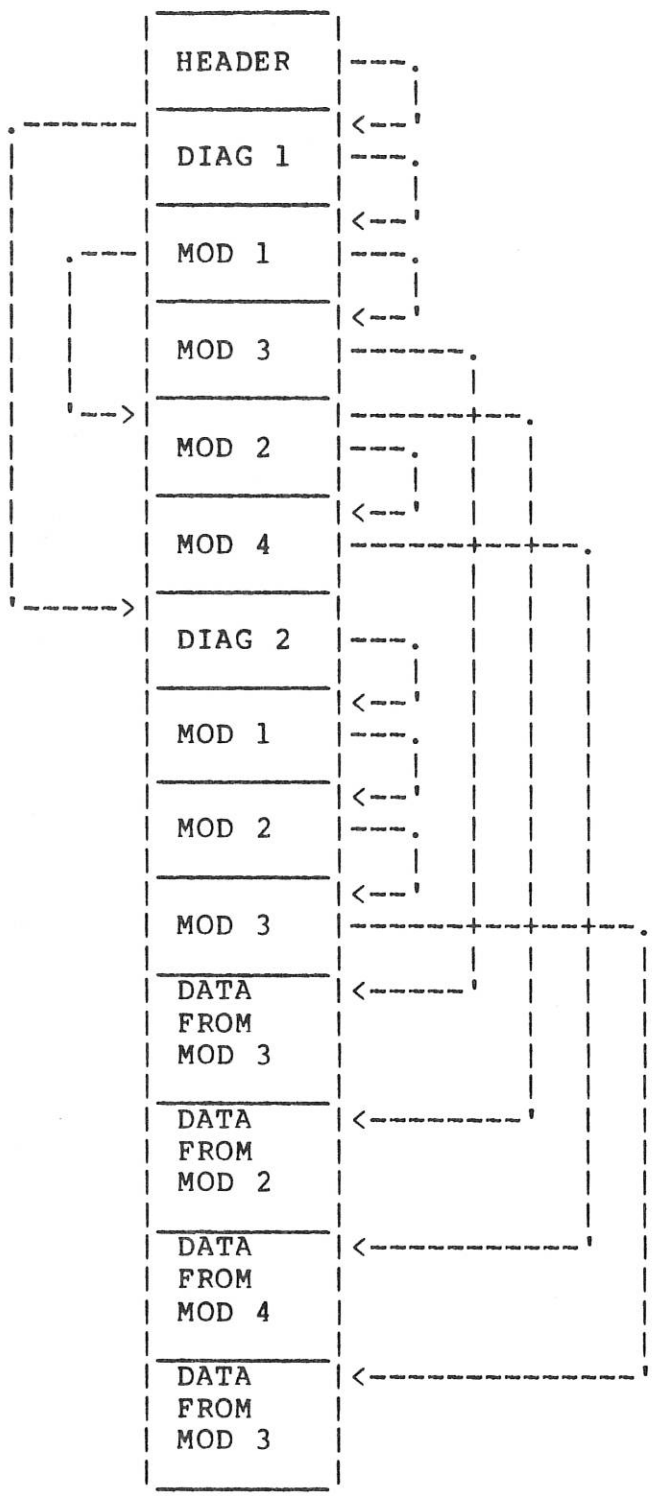


Figure 4-2
Data File Structure

CHAPTER 5

GALE System Generation

5.1 Introduction

The object of the system generation procedure is to create a GALE system tailored to the user's local hardware configuration and performance requirements. This section provides an outline of the stages involved in performing a system generation. Sysgen can be viewed as having three milestones, which are:

1. Generation of DIOS, the GALE Dynamic Input/Output System

2. Generation or copy of the GALE tasks and user object libraries.

3. Generation of the GALE Configuration structure model.

The entire system generation process, of which these phases are components, is directed by several indirect command files, wherein most of them are called by the main command file GALESGN.CMD. The procedure required for transforming the Configuration structure into a GALE compatible image form is contained in the file CNFGEN.CMD. The command files for installing DIOS and for setting up GALE for usage may be called directly, to conform to the particular user requirements.

5.2 Prerequisites

Several points which should be observed when generating a GALE system are outlined in the following sections.

5.2.1 Target System Prerequisites

The Installation under which GALE is to be run must meet the following requirements:

1. The hardware configuration must support at least a minimal RSX11M Version 3.0 mapped system.

2. If CAMAC support is desired, the hardware configuration must include one of the systems supported by DIOS.

3. When the RSX system was generated, the following questions must have been answered as follows:

```
* DO YOU WANT CHECKPOINTING?  [Y/N]:  Y
* DO YOU WANT MEMORY MANAGEMENT UNIT SUPPORT?  [Y/N]:  Y
* DO YOU WANT DYNAMIC MEMORY ALLOCATION SUPPORT?  [Y/N]:  Y
* DO YOU WANT AUTOMATIC MEMORY COMPACTION?  [Y/N]:  Y
* DO YOU WANT MULTI-USER PROTECTION SUPPORT?  [Y/N]:  Y
* DO YOU WANT LOADABLE DRIVER SUPPORT?  [Y/N]:  Y
* ARE YOU PLANNING TO INCLUDE A USER-WRITTEN DRIVER?  [Y/N]:  Y
* WHAT IS THE ADDRESS OF THE HIGHEST INTERRUPT VECTOR?  [O]:
```

The address given must be greater than or equal to the highest interrupt vector to be used by a DIOS module or CAMAC crate.

```
* DO YOU WANT AST SUPPORT?  [Y/N]:  Y
* DO YOU WANT SEND/RECEIVE DIRECTIVES?  [Y/N]:  Y
* DO YOU WANT THE USER ORIENTED TERMINAL DRIVER?  [Y/N]:  Y
```

If you want features of the terminal driver other than those which are provided by the standard user oriented terminal driver, you must answer the above question with no and the following four questions with yes when they appear.

```
* DO YOU WANT TASKS TO BE CHECKPOINTABLE DURING TERMINAL
    INPUT?  [Y/N]:  Y
* DO YOU WANT READ WITH NO ECHO SUPPORT?  [Y/N]:  Y
* DO YOU WANT READ AFTER PROMPT SUPPORT?  [Y/N]:  Y
* DO YOU WANT TRANSPARENT TERMINAL READ/WRITE
    SUPPORT?  [Y/N]:  Y
* DO YOU REQUIRE THE EXECUTIVE ROUTINE $GTWRD?  [Y/N]:  Y
* DO YOU REQUIRE THE EXECUTIVE ROUTINE $PTWRD?  [Y/N]:  Y
```

4. Within the executive region there must be a common partition declared as follows:

```
SET /MAIN=DAPAR:xxxx:15:COM
```

where xxxx is up to 763 (16K executive) or 1163 (20K executive).

5. The RSX system must provide at least one system-controlled partition to load the Message Output driver.

6. The RSX system must also provide a partition named "DAS-COM" with attribute "COM" and a length of at least 100(8) bytes.

7. The size of the RSX system pool must be at least 2K bytes after all desired tasks have been installed.

8. The "BIG" versions of the MACRO-11 Assembler and the Task Builder, "BIGMAC" and "BIGTKB", must be available under UIC [1,54].

5.2.2 Disk Space Prerequisites

The free block requirements for the Distribution and Target System devices naturally differ from those given in the Chapter "Generating a DIOS System" of the "DIOS I/O Operations" manual and depend on the particular actions taken. The following is a guideline, which may satisfy for the most installations:

Distribution device

ca. 50 blocks	non-contiguous

Target System device

ca. 200 blocks	contiguous a` 40 blocks
ca. 450 blocks	non-contiguous
ca. 50 blocks	contiguous per DIOS driver test task
96 blocks	contiguous for Plot Demo task
52 blocks	contiguous for data file I/O interface test task
33 blocks	contiguous for Namelist test task

If GALE is generated for a Target System foreign to the running system, the running system disk must be updated with the GALE macros and symbol definitions, which is normally performed by a DIOS generation for that disk.

The partition "DASCOM" (attribute "COM", length 100(8) bytes) must be installed and "BIGMAC" and "BIGTKB" must be available in the running system.

The Target System devices need not neccessarily be the same for a DIOS generation and generation of any other GALE components in a subsequent submission of the GALESGN.CMD file. This may be considered in cases where the DIOS Target System disk, which will be the system disk of the Target installation, lacks free blocks.

NOTE

The Target System device for a DIOS generation always must contain a bootable RSX11M Version 3.0 mapped system, generated to run on the target installation. The Target System device for all other GALE components may be any Files-11 device.

5.3 Distribution Device Contents

GALE is distributed on a single RK05 disk. The Distribution disk contains all source, object, task image, library and command files needed to perform a GALE sysgen. The following is a complete list of all files contained on a distribution disk.

UIC [1,200] contains command and ODL files:

GALESGN.CMD	Directs the GALE generation process
CNFGEN.CMD	Generate Configuration files in GALE format
ACQ.CMD	Generate the GALE Acquisition task
DLG.CMD	Generate the GALE Dialog task
DIOSGEN.CMD	Directs the DIOS generation process
DADRVGEN.CMD	Generate DIOS system-resident routines
CAMACGEN.CMD	Generate CAMAC crate configuration
LDRACPGEN.CMD	Generate the loader ACP
DRVACPGEN.CMD	Generate driver ACP's
MOACPGEN.CMD	Generate Message Output processor
DIOMOUGEN.CMD	Generate DIOS Mount processor
INSDIOS.CMD	Perform first-time installation of DIOS
PAR.CMD	Display target system partitions with VMR
DEV.CMD	Display target system device list with VMR
CLEANUP.CMD	Delete temporary files
DIOMOU.ODL	Overlay description for DIOS MOU processor

UIC [1,210] contains object files, libraries, task image and message files.

DADAT.OBJ	Device tables for DA0:
DADRV.OBJ	DIOS system-resident routines; DA: driver
DIOLNK.OBJ	Routine to link DA: into device list
MODRV.OBJ	Loadable driver for Message Output
DIOSYM.OBJ	Global definitions to include in SYSLIB
ACQ.OLB	Acquire task components
DLG.OLB	Dialog task components
NML.OLB	Namelist components
IOLIB.OLB	GALE command line input components
DASLIB.OLB	GALE data file I/O interface components
PTL.OLB	GALE run-time plot package
CAMLIB.BOR	CAMAC specific components (BORER system)
CAMLIB.ICP	CAMAC specific components (ICP-11 system)
CAMLIB.DUM	Dummy CAMAC specific components
LDRACP.OLB	Loader ACP components
DRVACP.OLB	Driver ACP components
MOACP.OLB	Message Output ACP components
DIOMOU.OLB	DIOS Mount processor components
MCR.OLB	Required for DIOS Mount processor
INI.OLB	Required for DIOS Mount processor
CNF.TSK	Configuration Builder task
PLT.TSK	Plot interpreter task
PLTDEM.TSK	Plot Demo task
TSTNML.TSK	Namelist test task
TSTGDI.TSK	GALE data file I/O interface test task
CMTEST.TSK	CM driver test task
CNTEST.TSK	CN driver test task
FKTEST.TSK	FK driver test task
MATEST.TSK	MA driver test task
MSTEST.TSK	MS driver test task
MXTEST.TSK	MX driver test task
PGTEST.TSK	PG driver test task
QDTEST.TSK	QD driver test task
TSTEST.TSK	TS driver test task
MOTFMT.MSG	System Message File for Message Output
USRFMT.MSG	GALE standard User Message File
USRFMT.STB	Symbol table of USRFMT.MSG for CNF task

UIC [1,220] contains MACRO-11 source code files.

CAMTBL.BOR	Define Borer crate tables
CAMTBL.ICP	Define ICP crate tables
USRFMT.MAC	GALE standard User Message File

UIC [30,12] contains macro source files

DIOSYM.MAC	DIOS symbol definitions
PDEMAC.MAC	GALE and other usefull macros
RSXMAC.DEC	Original RSXMAC.SML as delivered by DEC

5.4 Preparing for GALE System Generation

Before beginning to generate a GALE system, the user must be familiar with all of the respective documentation. In addition the following information concerning the target system should be available before starting the generation process:

1. Size of the executive region (16K or 20K).
2. Address of the highest interrupt vector.
3. Number of the 100-byte block on which partition "DAPAR" begins.
4. Number of terminals, including those marked "OFF-LINE".
5. Unibus addresses, vector area addresses and interrupt priorities of all CAMAC crate or branch controllers.
6. Unibus address, interrupt vector address and interrupt priority of the trigger device.

Because the Distribution and Target System disks are actively manipulated during the GALE system generation, the user should copy these disks prior to their actual use in the GALE system generation. The user should thoroughly prepare a configuration model of the target system acquisition hardware and note the types and number of modules, number of diagnostics (DDB's), amount of data, length of the longest and maximum number of NCB's associated with one control block.

Mount Distribution and Target System device both with write enabled. If the running system disk is not your Target System device, it must have at least the GALE macros in file RSXMAC.SML and the GALE symbol definitions resident in SYSLIB.OLB (this is always true, if a DIOS has been generated on the current system disk). Also a partition "DASCOM" of type "COM" and of 100(8) bytes in length must be present.

In a multi-user system, log on under a privileged UIC and start the GALE sygen process by typing

```
@ddu:[1,200]GALESGN
```

where ddu is the Distribution device.

The indirect command file will guide you through the generation process by printing comments and asking questions on the user terminal.

5.5 GALESGN.CMD File Details

This section describes the entire GALESGN.CMD indirect command file by adding descriptive text to the actual command file to clarify the SYSGEN process. The command file is presented in upper-case text, where descriptive text is presented in upper- and lower-case text.

```
;
; GALE SYSTEM GENERATION
; REVISION DATE: 12-JAN-78
;
;
; THE FOLLOWING SERIES OF QUESTIONS DEFINE THE CHARACTERIS-
; TICS OF THE GALE SYSTEM-RESIDENT ROUTINES.
; THE FORM OF THE ANSWERS DEPENDS ON THE QUESTIONS AS INDI-
; CATED:
;
; [S]:  THE RESPONSE MUST BE A 1 TO 16 CHARACTER STRING.
;       <CR> IS LEGAL ONLY IF EXPLICITLY STATED OR A DEFAULT
;       IS SPECIFIED.
; <XX>  INDICATES THE QUESTION HAS A DEFAULT RESPONSE, "XX".
;       TO SELECT THIS RESPONSE, TYPE <CR>.
;
; [N]:  INPUT AN OCTAL NUMBER (0 TO 377) OR A DECIMAL NUMBER
;       (0. TO 255. WITH DECIMAL POINT).
;       0 MAY BE ENTERED BY TYPING <CR>.
;
; [Y/N]: YES/NO QUESTION.  TYPE "Y" IF THE ANSWER IS YES.
;       TYPE "N" OR <CR> IF THE ANSWER IS NO.
;
* (TYPE <CR> TO CONTINUE) [S]:

        If you have read and understood the text above, type
        <CR> to continue indirect command file processing.
```

```
;
; ASSIGN DEVICES:
;
; "DISTRIBUTION DEVICE" IS THE DEVICE CONTAINING THIS COM-
; MAND FILE AND ALL OTHER COMMAND, OBJECT, AND MACRO FILES
; NEEDED TO GENERATE GALE.
;
; "TARGET SYSTEM DEVICE" IS THE DEVICE ON WHICH THE
; GALE SYSTEM BEING GENERATED IS TO RUN.
;
; "TARGET UIC" IS THE UIC UNDER WHICH THE OBJECT LIBRARY
; OR TASK WILL BE STORED ON THE TARGET SYSTEM DEVICE.
;
; "MAP DEVICE" IS THE DEVICE ON WHICH ALL TASK-BUILDER MAPS
; WILL BE GENERATED.
;
; "LIST DEVICE" IS THE DEVICE ON WHICH ALL ASSEMBLER
; LISTINGS WILL BE GENERATED.
;
* INPUT THE DISTRIBUTION DEVICE [DDU:] <DK1:> [S]:
```

Specify the device from which you have started this command file. If this device is DK1: you may simply type <CR>.

```
* INPUT THE TARGET SYSTEM DEVICE [DDU:] <SY0:> [S]:
```

Specify the device on which you want to generate the GALE system. If you intend to generate DIOS within this procedure, this device must contain a bootable RSX11M Version 3.0 mapped system. If your Target System device is the actual RSX11M system disk, you may simply type <CR>.

```
* INPUT THE TARGET UIC [[NN,MM]] <[1,100]> [S]:
```

Specify the UIC under which the GALE components should be generated on your target disk. For empty input this UIC defaults to [1,100].

```
* INPUT THE MAP DEVICE [DDU:] OR <CR> IF NO MAPS WANTED [S]:
* INPUT THE LIST DEVICE [DDU:] OR <CR> IF NO LISTINGS WANTED [S]:
```

Specify the devices where assembly listings and task build maps respectively should go. If no listings and/or maps are desired, reply with <CR>, which sets both device specifiers to NL:, the NULL device specifier.

```
* DO YOU WANT THE LISTINGS AND/OR MAPS TO BE SPOOLED? [Y/N]:
```

This question appears only if the print-spooler is installed and a list- and/or a map-device has been specified explicitly.

```
ASN 'Distribution Device'=DD:
ASN 'Target System Device'=TS:
ASN 'Map Device'=MP:
ASN 'List Device'=LS:
```

All logical devices are assigned the corresponding physical devices.

```
INS [1,54]PIP
INS [1,54]BIGMAC
INS [1,54]LBR
INS [1,54]BIGTKB
```

Installation is done only of those tasks, which are not yet installed.

```
;
; IF ONLY SOME COMPONENTS OF GALE ARE TO BE GENERATED
; THE FOLLOWING QUESTION SHOULD BE DENIED.
;
* IS THIS THE FIRST GALE GENERATION? [Y/N]:
```

If the answer is Y, the query/command sequence flagged with "/1/" is additionally performed. If the answer is N or <CR>, the query/command sequence flagged with "/2/" is additionally performed.

```
;                                                              /1/
; YOU MUST GENERATE DIOS BEFORE GENERATING THE GALE SYSTEM.    /1/
; A FULL DESCRIPTION OF HOW TO GENERATE DIOS IS GIVEN IN THE   /1/
; "DIOS I/O OPERATIONS" MANUAL. IF YOU ARE NOT YET FAMILIAR    /1/
; WITH THE DIOS SYSGEN PROCEDURE ANSWER THE FOLLOWING QUESTION /1/
; WITH "N" AND THEN RESTART THIS COMMAND FILE AFTER READING    /1/
; THE RESPECTIVE DOCUMENTATION.                                /1/
;                                                              /1/
* DO YOU WANT TO GENERATE DIOS NOW? [Y/N]:                     /1/
```

If the answer is N or <CR>, the GALE SYSGEN procedure will be terminated. The DIOS generation provides your target system disk with facilities you need in completing the GALE SYSGEN process. If the answer was Y, the DIOS generation procedure is invoked as descibed in Appendix E of the "DIOS I/O Operations" manual, where the initial questions concerning device and listing options are omitted.

```
* DO YOU WANT TO TERMINATE GALE GENERATION? [Y/N]:             /1/
```

If you want to generate DIOS and other GALE components on seperate devices, you can now terminate the generation process to start it over with new device specifications.
If the answer is Y, command file processing is terminated, otherwise the generation process goes on.

```
;                                                                    /2/
; IN THE FOLLOWING YOU MAY SPECIFY THE COMPONENTS                    /2/
; OF THE GALE SYSTEM, WHICH YOU WANT TO REBUILD.                     /2/
;                                                                    /2/
; THE ACQ TASK COLLECTS EXPERIMENTAL DATA AND STORES                 /2/
; THEM ON THE GALE DATA DEVICE.                                      /2/
; IF ANY TASK BUILD PARAMETERS DID NOT MEET YOUR                     /2/
; REQUIREMENTS, REBUILD ACQ. THE FORMER SPECIFICATIONS               /2/
; ARE AS FOLLOWS:                                                   /2/
;                                                                    /2/
PIP TI:=TS:'Target UIC'ACQ.BLD                                       /2/
* DO YOU NEED TO GENERATE THE ACQ TASK? [Y/N]:                       /2/
```

The former task build file is listed on the terminal. Check if the specified parameters meet your requirements. If some should be modified, answer the question with Y. If the answer is N, the query session concerning the ACQ task and the Trigger device is bypassed.
The following questions concerning the ACQ task are always expanded, if this is the first GALE generation.

```
;
; ACQUIRE TASK GENERATION
; REVISION DATE: 03-JAN-78
;
;
; THE ACQUIRE TASK REQUIRES AN MCB FOR YOUR SPECIFIC
; TRIGGER MODULE.
;
* HAVE YOU ALREADY DEFINED A TRIGGER MCB? [Y/N]:
```

If an MCB for your Trigger device has not yet been specified or the existing MCB does not meet your requirements, answer with N or <CR>. If the answer is Y, the following questions defining the Trigger MCB are omitted.

```
;
; IN ORDER TO SET UP AN TRIGGER MCB YOU HAVE TO SPECIFY
; THE TWO CHARACTER DRIVER ACP PREFIX WHICH CONTAINS
; THE TRIGGER DRIVER.
;
* INPUT DRIVER ACP PREFIX [S]:
```

Specify the driver ACP which contains the driver for the Trigger device (TGDRV). Driver ACP's are named xxACP, where the prefix xx is defined at DIOS generation time.

```
;
; YOU ALSO NEED TO SPECIFY THE TRIGGER INTERRUPT VECTOR,
; THE INTERRUPT PRIORITY AND UNIBUS CSR ADDRESS.
;
* INPUT TRIGGER INTERRUPT VECTOR ADDRESS [S]:
* INPUT TRIGGER INTERRUPT PRIORITY [S]:
* INPUT TRIGGER UNIBUS ADDRESS [S]:
```

GALE provides the Trigger device to be connected to a PDP11 via a DR11A or DR11C interface. Specify interrupt vector address, interrupt priority and Unibus address of the respective interface card, which completes the definition of the Trigger device MCB.

```
;
; IN ORDER TO GENERATE THE ACQ TASK YOU HAVE TO SPECIFY
; THE NUMBER OF LOGICAL UNIT NUMBERS TO BE USED.
; THIS IS A DECIMAL NUMBER WITHOUT DECIMAL POINT.
;
; THE NUMBER OF LUNS USED IS THE NUMBER OF MCB'S + 6.
;
* INPUT NUMBER OF LOGICAL UNITS <12> [S]:
```

Specify the number of LUN's for the ACQuire task. The minimum number is found by adding 6 to the number of MCB's which are contained in your GALE Configuration Structure model. A <CR> answer defaults the number of LUN's to 12, which allows a maximum of 6 modules, including the mandatory Trigger module, to be accessed via the ACQ task.

```
;
; THE ACQUIRE TASK CONTAINS FOUR INTERNAL BUFFERS WHICH MUST BE
; SPECIFIED AS AN OCTAL NUMBER.
;
; THE CNF BUFFER CONTAINS THE CONTROL STRUCTURE OF ALL
; DIAGNOSTICS WHICH ARE CURRENTLY DECLARED AS ON-LINE.
;
* INPUT LENGTH OF CNF BUFFER <4000> [S]:
```

Specify the length of the CNF buffer in bytes. The CNF buffer of the ACQ task is provided to hold all control blocks concerned with on-line data acquisition diagnostics. Adding the values given at offset B.LNG of all configuration control blocks and multiplying the sum by 100(8) gives the exact length of the CNF buffer. Note that the length may also be easily obtained from the assembly listing of the CNF structure model (see Chapter 2).

```
;
; THE DATA BUFFER MUST BE LARGE ENOUGH TO CONTAIN ONE BLOCK
; OF DATA (512 BYTES) FOR EACH MODULE WHICH DELIVERS DATA
; BEFORE OR DURING AN EXPERIMENT.
;
* INPUT LENGTH OF DATA BUFFER <2000> [S]:
```

Specify the length of the data buffer in bytes. ACQ uses this buffer for temporary storage of data read in from the on-line data acquisition diagnostic modules, before they are preserved on the GALE Data Device. The buffer must be large enough to hold all data requested from pre-shot modules (MC.PRE set) plus 8 bytes for each module. If the byte count per module exceeds 504 (512 respectively), 1024 bytes must be reserved for that module.
If only data from post-shot modules (MC.PST set) are taken a buffer of 512 bytes in length will satisfy.
Empty input <CR> defaults the data buffer to 2000(8) bytes in length.

```
;
; THE AST CONTROL BLOCK BUFFER MUST HAVE THE LENGTH OF THE
; NUMBER OF MODULES WHICH DELIVER DATA DURING AN EXPERIMENT * 20
;
* INPUT LENGTH OF AST CONTROL BLOCK BUFFER <100> [S]:
```

Specify the length of the AST control block buffer in bytes. This buffer contains control blocks for each module delivering data before or during an experiment. Control blocks are fixed to 16 bytes, where the length of the AST control block buffer is the number of pre-shot modules (MC.PRE set) times the length in bytes of a control block. Empty input allocates this buffer with 100(8) bytes, which is satisfactory for up to 4 pre-shot modules.

```
;
; THE I/O CONTROL BUFFER MUST HAVE THE LENGTH OF THE
; TOTAL NUMBER OF MODULES WHICH DELIVER DATA * 6.
;
* INPUT LENGTH OF I/O CONTROL BUFFER <74> [S]:
```

Specify the length of the I/O control buffer in bytes. This bufffer holds for each module delivering data an I/O control block of 6 bytes. Empty input defaults the buffer length to 74(8) bytes, which allows up to 10 data delivering modules.

```
;
; AFTER AN EXPERIMENT IS FINISHED, ACQUIRE WILL WAIT A PERIOD
; OF TIME TO ALLOW CURRENT I/O REQUESTS TO BE COMPLETED.
; AFTER THIS TIME ALL STILL OUTSTANDING REQUESTS WILL BE
; KILLED AND REQUESTS FOR POST EXPERIMENT DATA STARTED.
;
* INPUT DELAY TIME IN SECONDS <5> [S]:
```

Input the delay time, which requests to pre-shot modules can overdue the end-of-experiment trigger. The specifications of the ACQ task are now finished. Thereafter the session concerning the DLG task is entered. If this is the first GALE generation the lines flagged with "/2/" are omitted and the following query session concerning the DLG task is always performed.

```
;                                                              /2/
; THE DLG TASK PROVIDES DYNAMIC MODIFICATIONS                   /2/
; OF THE GALE DATA BASE. IF ANY TASK BUILD                      /2/
; PARAMETERS DID NOT MEET YOUR REQUIREMENTS,                    /2/
; REBUILD DLG. THE FORMER SPECIFICATIONS ARE AS FOLLOWS:        /2/
;                                                              /2/
PIP TI:=TS:'Target UIC'DLG.BLD                                 /2/
* DO YOU NEED TO GENERATE THE DLG TASK? [Y/N]:                  /2/
```

The former task build file is listed on the terminal. Check if the specified parameters meet your requirements. If some should be modified, answer the question with Y. If the answer is N, the query session concerning the DLG task is bypassed.

```
;
; DIALOG GENERATION
; REVISION DATE: 03-JAN-78
;
;
; IN ORDER TO GENERATE THE DLG TASK YOU HAVE TO SPECIFY THE SIZE
; OF THE INTERNAL BUFFERS.
; THE BUFFER SIZE SPECIFICATION IS AN OCTAL NUMBER.
;
; THE DIRECTORY BUFFER MUST HAVE THE SIZE 36 * MAX. NUMBER OF DDB'S.
;
* INPUT LENGTH OF DIRECTORY BUFFER <74> [S]:
```

The directory buffer holds for each DDB a control table of a fixed length of 36(8) bytes. Therefore the length of the directory buffer is given as 36(8) times the number of DDB's contained in the GALE Configuration File. Empty input allocates a directory buffer for two DDB's.

```
;
; THE BLOCK BUFFER MUST HAVE THE SIZE OF THE LONGEST CONTROL BLOCK
;
* INPUT LENGTH OF LONGEST CONTROL BLOCK BUFFER <300> [S]:
```

Specify the length of the longest control block contained in the GALE Configuration File. The length should be a multiple of 100(8), as GALE configuration control blocks are allocated in records of 100(8) bytes (see entry B.LNG). Empty input defaults the buffer length to 300(8), which is satisfactory for control blocks consisting of up to 3 records.

```
;
; THE NAMELIST BUFFER MUST HAVE THE SIZE OF THE MAX. NUMBER
; OF NAMELIST ELEMENTS FOR ONE CONTROL BLOCK * 20.
;
* INPUT LENGTH OF THE NAMELIST BUFFER <600> [S]:
```

Choose the length of the namelist buffer so that all NCBD's concerning to one control block may fit in. Empty input allocates the buffer for a maximum of 24 NCBD's.
The DLG task build parameters are now complete. As mentioned above, the following lines flagged with "/2/" are expanded only, if this is not the first GALE generation.

```
;                                                              /2/
; GALE CONFIGURATION BUILDER TASK                              /2/
;                                                              /2/
* DO YOU NEED TO COPY THE CNF TASK? [Y/N]:                     /2/
;                                                              /2/
; GALE DATA FILE I/O INTERFACE TEST TASK                       /2/
;                                                              /2/
* DO YOU NEED TO COPY THE TSTGDI TASK? [Y/N]:                  /2/
;                                                              /2/
; NAMELIST TEST TASK                                           /2/
;                                                              /2/
* DO YOU NEED TO COPY THE TSTNML TASK? [Y/N]:                  /2/
;                                                              /2/
; GALE PLOT INTERPRETER TASK                                   /2/
;                                                              /2/
* DO YOU NEED TO COPY THE PLT TASK? [Y/N]:                     /2/
* DO YOU NEED TO COPY THE PLOT DEMO TASK? [Y/N]:               /2/
* DO YOU NEED TO COPY GALE USER OBJECT LIBRARIES? [Y/N]:       /2/
```

Each question answered with Y causes the respective files, task image or object libraries, to be deleted on your Target System device under the specified Target UIC and to be transferred newly from the Distribution device to your Target System device.

```
;
; IN ORDER TO GENERATE THE GALE TASKS THE COMMON BLOCK
; "DASCOM" MUST BE GENERATED.
;
* DO YOU NEED TO GENERATE THE COMMON BLOCK "DASCOM"? [Y/N]:     /2/
```

Answer with Y, if the existing "DASCOM" should be replaced by a new version. If the answer is N, the following questions concerning the common block "DASCOM" are bypassed.

```
;
; EACH EXPERIMENT IS KNOWN BY A 3-CHARACTER SHORT NAME.
;
* WHAT IS YOUR EXPERIMENT SHORT NAME [S]:
```

Input the 3-character short name. This name should be unique to your application in your local area, as it identifies your GALE system.

```
;
; ALL GALE DATA FILES ARE STORED UNDER A SPECIFIC UIC.
; SIMILARILY THE SYTEM FILES ARE ALSO STORED UNDER ONE UIC,
; WHICH MAY BE DIFFERENT FROM THE DATA FILE UIC.
;
* INPUT THE GALE DATA FILE UIC <[1,100]> [S]:
* INPUT THE GALE SYSTEM FILE UIC <[1,100]> [S]:
```

Input the respective UIC's as indicated. The GALE data file UIC specifies the UIC under which the data files, generated by the ACQ task, are to be stored. The GALE system file UIC specifies the UIC under which The GALE Configuration File with corresponding Namelist File are expected by the different GALE tasks.
The following lines are self-explanatory and serve to copy DIOS driver test tasks onto your Target System device. They are repeated until an empty input for a device identifier is given.

```
;
; EACH OF THE DIOS DRIVER TEST TASKS WILL BE COPIED TO YOUR
; TARGET DEVICE IN TURN. THE FOLLOWING DIOS DRIVERS CURRENTLY
; HAVE ASSOCIATED TEST TASKS.
;
; CM      CAMAC MEMORY
; CN      NUCLEAR ENTERPRISES CAMAC MEMORY
; FK      FUNCTION KEYBOARD
; MA      MULTI-CHANNEL ANALYZER
; MS      MULTI-SCALER
; MX      CAMAC MULTIPLEXER
; PG      CAMAC PULSE GENERATOR
; QD      QD808 CHARGE DIGITIZER
; TS      CULHAM TIME SEQUENCE GENERATOR
;
; TO COPY A TEST TASK, ENTER THE 2-CHARACTER IDENTIFIER IN REPLY
; TO THE FOLLOWING QUESTION. AN INPUT OF <CR> INDICATES THAT
; NO MORE DRIVER TEST TASKS ARE TO BE COPIED.
;
* INPUT THE DEVICE IDENTIFIER [S]:
PIP TS:'Target UIC''Device Identifier'TEST.TSK;*/DE
PIP TS:'Target UIC'=DD:[1,210]'Device Identifier'TEST.TSK

        Now all required DIOS driver test  tasks  have  been
        copied.
        If this is the first GALE Sysgen all following  com-
        mand  lines  are executed.  Otherwise only those are
        invoked which have been selected explicitly  in  the
        query session above.

MAC DD:[1,210]DASCOM,LS:'$SP'=DD:[1,220]DASCOM
PIP TS:[1,1]DASCOM.*;*/DE
TKB @DD:[1,200]DASCOM.BLD
PIP DD:[1,200]DASCOM.BLD;*/DE
PIP DD:[1,210]DASCOM.OBJ;*/DE
PIP DD:[1,220]DASCOM.MAC/PU
PIP TS:'Target UIC'NML.OLB;*/DE,IOLIB.OLB;*,DASLIB.OLB;*,PTL.OLB;*
PIP TS:'Target UIC'=DD:[1,210]NML.OLB,IOLIB.OLB,DASLIB.OLB,PTL.OLB
PIP TS:'Target UIC'CNF.TSK;*/DE
PIP TS:'Target UIC'=DD:[1,210]CNF.TSK
PIP TS:'Target UIC'=DD:[1,210]DLG.OLB
;
; DIALOG GENERATION
; REVISION DATE: 03-JAN-78
;
PIP TS:'Target UIC'DLG.TSK;*/DE
TKB @TS:'Target UIC'DLG.BLD
PIP TS:'Target UIC'DLG.BLD/PU
PIP TS:'Target UIC'DLG.OLB;*/DE
PIP TS:'Target UIC'=DD:[1,210]ACQ.OLB
```

```
;
; ACQUIRE TASK GENERATION
; REVISION DATE: 03-JAN-78
;
MAC DD:[1,210]TGMCB,LS:'$SP'=TS:'$UIC'TGMCB
LBR TS:'$UIC'ACQ/RP=DD:[1,210]TGMCB
PIP DD:[1,210]TGMCB.OBJ;*/DE
PIP TS:'$UIC'TGMCB.MAC/PU
PIP TS:'$UIC'ACQ.TSK;*/DE
TKB @TS:'$UIC'ACQ.BLD
PIP TS:'$UIC'ACQ.BLD;*/PU
PIP TS:'Target UIC'ACQ.OLB;*/DE
PIP TS:'Target UIC'PLT.TSK;*/DE
PIP TS:'Target UIC'=DD:[1,210]PLT.TSK
PIP TS:'Target UIC'PLTDEM.TSK;*/DE
PIP TS:'Target UIC'=DD:[1,210]PLTDEM.TSK
PIP TS:'Target UIC'TSTGDI.TSK;*/DE
PIP TS:'Target UIC'=DD:[1,210]TSTGDI.TSK
PIP TS:'Target UIC'TSTNML.TSK;*/DE
PIP TS:'Target UIC'=DD:[1,210]TSTNML.TSK
```

At this point all GALE components are built. The initially installed RSX utility tasks are now removed.

```
REM ...PIP
REM ...MAC
REM ...LBR
REM ...TKB
;
; YOUR GALE SYSTEM IS NOW COMPLETE. IF YOU HAVE NOT YET
; PREPARED A DESCRIPTION MODEL OF YOUR RELEVANT
; ACQUISITION HARDWARE, YOU SHOULD DO THIS NEXT.
; FOR FURTHER INFORMATION REFER TO CHAPTER "GENERATION
; OF GALE CONFIGURATION FILES" OF THE "GALE SYSTEM
; PROGRAMMER'S HANDBOOK". ONCE YOU HAVE PREPARED
; THE RESPECTIVE SOURCE FILES, START THE INDIRECT
; COMMAND FILE "CNFGEN.CMD". THIS BATCH IS ON YOUR
; DISTRIBUTION DEVICE AND WILL GUIDE YOU THROUGH
; THE FURTHER CONFIGURATION BUILD PROCESS.
;
@ <EOF>
```

5.6 Building GALE Configuration Files

The GALE system is based upon a Configuration file, which gives a standarized description of the relevant data acquisition hardware. Once the user has prepared the Configuration and associated Namelist source files, as outlined in Chapter 2, he may start transforming the files for use with GALE, provided the CNF (Configuration File Builder) task is present on the same device under the same UIC where the source files have been prepared. Also the GALE macro and symbol definitions must be present in files RSXMAC.SML and SYSLIB.OLB under UIC [1,1] on the current system disk.

5.6.1 CNFGEN.CMD File Details

This section describes the entire CNFGEN.CMD indirect command file by adding descriptive text to the actual command file to clarify the CNF-Generation process. The command file is presented in upper-case text, whereas descriptive text is presented in upper- and lower-case text.

```
;
; EXPERIMENT CNF STRUCTURE FILE GENERATION
; REVISION DATE: 05-JAN-78
;
;
; THE FOLLOWING SERIES OF QUESTIONS DEFINE THE CHARACTERIS-
; TICS OF THE CNF STRUCTURE GENERATION PROCESS.
; THE FORM OF THE ANSWERS DEPENDS ON THE QUESTION, AS INDI-
; CATED:
;
; [S]:  THE RESPONSE MUST BE A 1 TO 16 CHARACTER STRING.
;       <CR> IS LEGAL ONLY IF EXPLICITLY STATED OR A DEFAULT
;       IS SPECIFIED.
; <XX>  INDICATES THE QUESTION HAS A DEFAULT RESPONSE, "XX".
;       TO SELECT THIS RESPONSE, TYPE <CR>.
;
; [N]:  INPUT AN OCTAL NUMBER (0 TO 377) OR A DECIMAL NUMBER
;       (0. TO 255. WITH DECIMAL POINT).
;       0 MAY BE ENTERED BY TYPING <CR>.
;
; [Y/N]: YES/NO QUESTION.  TYPE "Y" IF THE ANSWER IS YES.
;       TYPE "N" OR <CR> IF THE ANSWER IS NO.
;
* (TYPE <CR> TO CONTINUE) [S]:

        If you have read and understood the text above, type
        <CR> to continue indirect command file processing.
```

```
;
; ASSIGN DEVICES:
;
; "DISTRIBUTION DEVICE" IS THE DEVICE CONTAINING THIS COM-
; MAND FILE.
;
; "TARGET SYSTEM DEVICE" IS THE DEVICE ON WHICH THE
; GALE SYSTEM WAS GENERATED.
;
; "TARGET UIC" IS THE UIC UNDER WHICH THE CONFIGURATION
; IMAGE FILE WILL BE STORED ON THE TARGET SYSTEM DEVICE.
;
; "MAP DEVICE" IS THE DEVICE ON WHICH ALL TASK-BUILDER MAPS
; WILL BE GENERATED.
;
; "LIST DEVICE" IS THE DEVICE ON WHICH ALL ASSEMBLER
; LISTINGS WILL BE GENERATED.
;
* INPUT THE DISTRIBUTION DEVICE [DDU:] <DK1:> [S]:
* INPUT THE TARGET SYSTEM DEVICE [DDU:] <SY0:> [S]:
* INPUT THE TARGET UIC [[NN,MM]] <[1,100]> [S]:
```

Specify devices and UIC as you have done at GALE Sysgen time. The Target System device must contain the CNF task and the Configuration source file with corresponding Namelist source file under the Target UIC. Also the common block "DASCOM" for your GALE system must be available on the Target System device under UIC [1,1].

```
* INPUT THE MAP DEVICE [DDU:] OR <CR> IF NO MAPS WANTED [S]:
* INPUT THE LIST DEVICE [DDU:] OR <CR> IF NO LISTINGS WANTED [S]:
```

Specify list and map device or give empty input if no listings/maps desired.

```
* DO YOU WANT THE LISTINGS AND/OR MAPS TO BE SPOOLED? [Y/N]:
```

This question appears only if the print-spooler is installed and a list and/or map device has been specified explicitly.

```
ASN 'Distribution Device'=DD:
ASN 'Target System Dveice'=TS:
ASN 'Map Device'=MP:
ASN 'List Device'=LS:
```

All logical devices are assigned the corresponding physical devices.

```
INS [1,54]BIGMAC
INS [1,54]PIP
INS [1,54]BIGTKB
```

Installation is done of those tasks only, which are not yet installed.

```
;
; IN ORDER TO GENERATE THE CONFIGURATION STRUCTURE FILES FOR YOUR
; SYSTEM YOU MUST HAVE SOURCE FILES PREPARED AS DESCRIBED IN
; THE "GALE SYSTEM PROGRAMMER'S HANDBOOK".
; THE SOURCE FILE FOR THE CONFIGURATION STRUCTURE SHOULD HAVE BEEN
; NAMED "EXP"CNF.MAC AND YOUR SOURCE FILE FOR THE CORRESPONDING
; NAMELIST STRUCTURE SHOULD HAVE BEEN NAMED "EXP"NCB.MAC,
; WHERE "EXP" IS THE 3-CHARACTER EXPERIMENT SHORT NAME
; WHICH YOU GAVE IN DEFINING DASCOM IN THE GALE SYSTEM GENERATION.
;
* HAVE YOU ALREADY PREPARED YOUR SOURCE FILES? [Y/N]:
```

If you have prepared the Configuration source file along with the corresponding Namelist source file under the Target UIC on your Target System device and both files have been named as outlined above, you may answer this question with Y. In all other cases you should reply with N or <CR>, which causes processing of this indirect command file to be terminated.

```
* WHAT IS YOUR EXPERIMENT SHORT NAME [S]:
```

Input the 3-character short name of your experiment as you specified it at GALE Sysgen time.

```
;
; THE TARGET SYSTEM DASCOM MUST BE INSTALLED IN ORDER TO GENERATE
; THE CONFIGURATION STRUCTURE FILES. THIS SHOULD BE DONE IF NO
; GALE USER'S ARE ACTIVE, ESPECIALLY IF YOU ARE GENERATING FOR
; A TARGET SYSTEM FOREIGN TO THE CURRENT SYSTEM.
;
* MAY THE TARGET SYSTEM DASCOM MAY BE INSTALLED NOW? [Y/N]:
```

If the answer is Y, the common block "DASCOM" is installed from the Target System device, thus overwriting a formerly installed "DASCOM". This may lead to a malfunction of GALE tasks and even to disruption of GALE components. To avoid any errors ensure that no GALE users are active at present, otherwise answer with N or <CR>, which terminates indirect command file processing, and try CNF file generation later.
If you are generating for a Target System foreign to the current system, don't forget to reinstall "DASCOM" from the current system disk, if it was installed formerly.

```
INS TS:[1,1]DASCOM
INS TS:'Target UIC'CNF
```

The common block "DASCOM" is installed from the Target System device. If the CNF task is not already installed, it will be performed now. If it cannot be installed from the specified Target System device the following lines will appear on your terminal and command file processing will be terminated.

```
;
; YOUR TARGET SYSTEM DEVICE CONTAINS NO CNF BUILDER TASK-
; IMAGE FILE. CORRECT THE MISTAKE AND START CNFGEN.CMD OVER.
;
```

If installation of the CNF task was successful, command file processing is continued here.

```
;
; THE USER MESSAGE FORMAT FILE, USRFMT.MSG, PROVIDES EXPLANATORY
; INFORMATION ABOUT THE CONTENTS OF AN NCB. THE STANDARD FILE
; HAS ALREADY BEEN COPIED TO YOUR TARGET SYSTEM DISK IN THE
; GALE SYSTEM GENERATION PROCEDURE. THE GALE SYSTEM PROGRAMMER'S
; HANDBOOK PROVIDES INFORMATION ON HOW TO EXTEND THE CONTENTS
; OF THE FILE. IF YOU HAVE CHANGED THE CONTENTS OF THE USER
; MESSAGE FORMAT FILE YOU MUST NOW REBUILD IT. AFTER REBUILDING
; THE FILE YOU MUST DISMOUNT AND THEN REMOUNT MO: IN ORDER TO
; MAKE THE NEW MESSAGE FILE EFFECTIVE.
;
* DO YOU WANT TO REBUILD USRFMT.MSG? [Y/N]:
```

The source file of the standard user messages was copied to your Target System device at GALE Sysgen time and resides under UIC [1,2]. If you have changed the contents of the file and want to make the new version available, answer with Y, otherwise input <CR> or N.

```
MAC DD:[1,210]USRFMT,LS:'Spool Attribute'=TS:[1,2]USRFMT
TKB @DD:[1,200]USRFMT.BLD
CNF TS:[1,2]USRFMT.MSG/CF=DD:[1,210]USRFMT
PIP DD:[1,200]USRFMT.BLD;*/DE
PIP DD:[1,210]USRFMT.OBJ;*/DE,USRFMT.TSK;*
```

If the user has decided to rebuild the User Message File it is done via the steps outlined above. First the source file is assembled, then a task image is built from which the pure binary version is produced by the GALE utility CNF. Afterwards all temporary files are deleted.

Whether or not the User Message File was to be rebuilt, indirect command file processing continues at this point.

```
MAC DD:[1,210]DASNCB,LS:'Spool Attribute'=TS:'Target UIC''EXP'NCB
MAC DD:[1,210]DASCNF,LS:'Spool Attribute'=TS:'Target UIC''EXP'CNF
TKB @DD:[1,200]'EXP'NCB.BLD
TKB @DD:[1,200]'EXP'CNF.BLD
CNF TS:'Target UIC'DASNCB.'EXP'/NM=DD:[1,210]DASNCB
CNF TS:'Target UIC'DASCNF.'EXP'/CF=DD:[1,210]DASCNF
PIP DD:[1,200]'EXP'CNF.BLD;*/DE,'EXP'NCB.BLD;*
PIP DD:[1,210]DASCNF.*;*/DE,DASNCB.*;*
PIP DD:[1,210]*.STB/PU
```

The Configuration files are built as outlined above with the User Message File. All intermediate files, which are no longer used are deleted, where the last version of the Symbol Table for the User Message File is kept on the Distribution device for later generation of CNF files.

```
REM ...MAC
REM ...TKB
REM ...CNF
PIP TS:'Target UIC'SETUP.CMD;*/DE
REM ...PIP
```

All tasks installed during processing of this indirect command file are now removed. Existing versions of the indirect command file SETUP.CMD, which prepares the GALE system for usage, are replaced by a new version reflecting the present specifications.

```
;
; YOUR CONFIGURATION STRUCTURE MODEL IS NOW READY FOR USE.
;
* DO YOU WANT TO SET UP GALE FOR DATA ACQUISITION NOW? [Y/N]:
```

If the current system disk is your Target System device and you want to set up GALE for data acquisition, you may answer Y, which invokes the indirect command file SETUP.CMD just generated. If the answer is N or <CR> command file processing ends with the output of the following comment lines.

```
;
; TO SET UP GALE FOR DATA ACQUISITION YOU HAVE TO RUN
; THE INDIRECT COMMAND FILE "SETUP.CMD", WHICH WAS
; CREATED ON YOUR SYSTEM TARGET DISK UNDER YOUR TARGET UIC.
;
@ <EOF>
```

5.7 Setting up GALE for Usage

If you have successfully performed a complete GALE Sysgen, you may set it up for usage in either of two ways: If the question

```
* DO YOU WANT TO SET UP GALE FOR DATA ACQUISITION NOW? [Y/N]:
```

in the Configuration file build procedure is answered Y, or by invoking the indirect command file SETUP.CMD explicitly via MCR:

```
@TS:'Target UIC'SETUP
```

NOTE

Be sure that the running system disk contains a GALE Dynamic Input/Output System tailored to your requirements.

5.7.1 SETUP.CMD File Details

The following describes the entire SETUP.CMD indirect command file by adding descriptive text to the actual command file as it has been generated by the CNF file generation process. The command file is presented in upper-case text, where descriptive text is presented in upper- and lower-case text.

```
;
; SET UP GALE FOR DATA ACQUISITION
;
```

The following lines concerning the Target System device and the Target UIC are bypassed if setting up of the GALE system was invoked by the Configuration build process.

```
;
; IN ORDER TO INSTALL THE GALE TASKS YOU HAVE TO SPECIFY
; THE TARGET SYSTEM DEVICE AND TARGET UIC
; WHERE THE TASKS HAVE BEEN BUILT.
;
* INPUT THE TARGET SYSTEM DEVICE [DDU:] <SY0:> [S]:
* INPUT THE TARGET UIC [[NN,MM]] <[1,100]> [S]:
```

Specify the Target System device and UIC as you have done at GALE Sysgen and CNF generation time.

```
ASN 'Target System device'=TS:
```

The logical device is now assigned the physical device.

```
;
; IN ORDER TO ACTIVATE THE GALE SYSTEM YOU
; HAVE TO ACTIVATE THE DYNAMIC INPUT/OUPUT SYSTEM
; INCLUDING THE MESSAGE OUTPUT PROCESSOR.
;
* DO YOU WANT TO INSTALL DIOS NOW? [Y/N]:
```

If you answer Y, your Target System device must be the running system disk and assigned SY0:. The indirect command file to install the GALE Dynamic Input/Output System is then activated. For details refer to the "DIOS I/O OPERATIONS" manual.
Whether DIOS is to be installed or not, command file processing continues at this point.

```
;
; WHEN INSTALLING THE GALE ACQUISITION OR DIALOG TASK,
; THE COMMON BLOCK "DASCOM" MUST HAVE BEEN INSTALLED BEFORE.
;
* DO YOU WANT TO INSTALL THE COMMON BLOCK "DASCOM"? [Y/N]:
```

If the answer is Y, the common block "DASCOM" is installed from your Target System device. Note, that a formerly installed "DASCOM" is thereby overwritten.

```
* DO YOU WANT TO INSTALL THE GALE ACQUISITION TASK "ACQ"? [Y/N]:
* DO YOU WANT TO INSTALL THE GALE DIALOG TASK "DLG"? [Y/N]:
* DO YOU WANT TO INSTALL THE GALE PLOT INTERPRETER "PLT"? [Y/N]:
```

Each question answered with Y causes the corresponding task to be installed later on.
The following lines concerning the GALE data device and the GALE system device are bypassed if both, ACQ and DLG task are not to be installed.

```
;
; NOW SPECIFY THE GALE DATA DEVICE WHERE ALL GALE DATA FILES
; ARE TO BE STORED AND THE GALE SYSTEM DEVICE WHERE ALL OTHER
; GALE FILES RESIDE.
;
* INPUT GALE DATA DEVICE [DDU:] <DK1:> [S]:
* INPUT GALE SYSTEM DEVICE [DDU:] <SY0:> [S]:
```

Specify the GALE data device, where GALE data files are to be stored and the GALE system device, which is the device on which your Configuration structure files have been built. In the most cases the GALE system device will therefore be identical to your Target System device. If empty input is given the devices default to DK1: and SY0: respectively.

```
ASN 'GALE Data device'=DD:/GBL
ASN 'GALE System device'=DS:/GBL
```

The logical devices are assigned physical devices.

```
INS TS:[1,1]DASCOM
```

The common block "DASCOM" is installed, if it was stated above.
The following actions are taken only if ACQ or DLG task respectively should be installed.

```
REM ...ACQ
```

ACQ is removed if it had been installed, but is not active at present.

```
; CANNOT REMOVE/INSTALL -- ACQ IS ACTIVE
```

This message appears if ACQ is active when trying to install it.

```
INS TS:'Target UIC'ACQ
```

Otherwise ACQ is installed.

```
REM ...DLG
; CANNOT REMOVE/INSTALL -- DLG IS ACTIVE
INS TS:'Target UIC'DLG
REM ...PLT
; CANNOT REMOVE/INSTALL -- PLT IS ACTIVE
INS TS:'Target UIC'PLT
```

The installation procedure explained for the ACQ task is repeated for the DLG and PLT tasks.
Finally the User File Directory (UFD), where the GALE Plot intermediate data files are stored, is established.

```
INS [1,54]UFD
UFD TS:[1,7]
REM ...UFD
```

Don't forget to start the PLT task before producing any plots.

```
;
; GALE IS NOW READY TO RUN. GOOD LUCK!!!
;
@ <EOF>
```

APPENDIX A

Offset Definitions for CNF Blocks

Define paramter offsets relative to the base of the parameter area. TYP denotes "H" for Header, "D" for DDB and "M" for MCB. TYP1 denotes "UPR" for Header or DDB, and "DDP" for MCB.

```
        .MACRO  UPDEF   TYP,TYP1
TYP'.P01 = TYP'.'TYP1
TYP'.P02 = TYP'.'TYP1+2
TYP'.P03 = TYP'.'TYP1+4
TYP'.P04 = TYP'.'TYP1+6
TYP'.P05 = TYP'.'TYP1+10
TYP'.P06 = TYP'.'TYP1+12
TYP'.P07 = TYP'.'TYP1+14
TYP'.P08 = TYP'.'TYP1+16
TYP'.P09 = TYP'.'TYP1+20
TYP'.P10 = TYP'.'TYP1+22
TYP'.P11 = TYP'.'TYP1+24
TYP'.P12 = TYP'.'TYP1+26
TYP'.P13 = TYP'.'TYP1+30
TYP'.P14 = TYP'.'TYP1+32
TYP'.P15 = TYP'.'TYP1+34
TYP'.P16 = TYP'.'TYP1+36
TYP'.P17 = TYP'.'TYP1+40
TYP'.P18 = TYP'.'TYP1+42
TYP'.P19 = TYP'.'TYP1+44
TYP'.P20 = TYP'.'TYP1+46
TYP'.P21 = TYP'.'TYP1+50
TYP'.P22 = TYP'.'TYP1+52
TYP'.P23 = TYP'.'TYP1+54
TYP'.P24 = TYP'.'TYP1+56
TYP'.P25 = TYP'.'TYP1+60
TYP'.P26 = TYP'.'TYP1+62
TYP'.P27 = TYP'.'TYP1+64
TYP'.P28 = TYP'.'TYP1+66
TYP'.P29 = TYP'.'TYP1+70
TYP'.P30 = TYP'.'TYP1+72
TYP'.P31 = TYP'.'TYP1+74
TYP'.P32 = TYP'.'TYP1+76
TYP'.P33 = TYP'.'TYP1+100
TYP'.P34 = TYP'.'TYP1+102
TYP'.P35 = TYP'.'TYP1+104
TYP'.P36 = TYP'.'TYP1+106
TYP'.P37 = TYP'.'TYP1+110
TYP'.P38 = TYP'.'TYP1+112
TYP'.P39 = TYP'.'TYP1+114
TYP'.P40 = TYP'.'TYP1+116
TYP'.P41 = TYP'.'TYP1+120
TYP'.P42 = TYP'.'TYP1+122
TYP'.P43 = TYP'.'TYP1+124
        .ENDM   UPDEF
```

```
        .MACRO  BLKDF$
;
; DEFINE BLOCK TYPES AND GENERAL BLOCK OFFSETS
;
BT.HED = 1
BT.DDB = 2
BT.MCB = 3
BT.DAT = 4
B.LNG = 0
B.TYP = 2
B.DID = 3
B.LK1 = 4
B.LK2 = 6
B.NCB = 10
;
; DEFINE HEADER OFFSETS
;
H.FPC = 3
H.DAT = 12
H.DAY = 12
H.MON = 13
H.YEA = 14
H.TIM = 16
H.HOU = 16
H.MIN = 17
H.SEC = 20
H.LST = 22
H.NXT = 24
H.EXP = 26
H.FLN = 32
H.DBM = 34
H.LUP = 46
H.UPR = 50
;
; DEFINE DDB OFFSETS, STATUS AND CONTROL BITS
;
D.DID = 3
D.TYP = 12
D.MLN = 13
D.MSG = 14
D.UIC = 42
D.UOC = 42
D.UGC = 43
D.CHR = 44
D.STS = 45
D.RSA = 46
D.LUP = 56
D.UPR = 60
DC.DAQ = 1
DC.CTL = 2
DC.LCL = 100
DC.RMT = 200
ST.ONL = 1
```

```
;
; DEFINE MCB OFFSETS
;
M.DID = 3
M.TYP = 12
M.UNIT = 14
M.MID = 15
M.ACP = 16
M.CTL = 20
M.DLN = 22
M.DFM = 23
M.DCT = 24
M.DAT = 26
M.VCT = 30
M.PRI = 31
M.ADR = 32
M.ERR = 34
M.LUN = 35
M.LPM = 46
M.DDP = 50
;
; DEFINE MCB STATUS AND CONTROL BITS
;
MC.CTL = 1
MC.GEN = 2
MC.PRE = 10
MC.PST = 20
MC.HLD = 40
MC.INT = 100
MC.CAM = 200
MC.SUB = 400
MC.NPR = 4000
MC.OVF = 40000
MC.ERR = 100000
MF.NST = 0
MF.BYT = 1
MF.INT = 2
MF.FLT = 3
MF.MUX = 4
MF.NL1 = 10
MF.NL2 = 20
;
; DEFINE DATA BLOCK OFFSETS
;
X.MID = 4
X.BLN = 6
X.DAT = 10
;
; DEFINE UPAR TYPE
;
U.ASC = 0
U.BYT = 1
U.INT = 2
U.FLT = 4
```

```
;
; DEFINE NCBD ACCESS FLAGS
;
NS.PRV = -1
NS.NPR = 0
NS.NDF = 0
;
; DEFINE NCB AND NCBD OFFSETS
;
N.PRV = -2
N.NXT = 0
N.HLP = 2
N.NAM = 4
N.FLG = 10
N.TYP = 11
N.LEN = 12
N.MSK = 13
N.OFS = 14
;
; DEFINE NCB AND NCBD DATA FLAGS
;
NS.LCL = 0
NS.RMT = 1
NS.HLP = 2
;
; DEFINE USER PARAMETER OFFSETS
;
        .MCALL  UPDEF
UPDEF   H,UPR
UPDEF   D,UPR
UPDEF   M,DDP
        .ENDM   BLKDF$
;
; DEFINE ALL OF ABOVE MACROS AND OFFSETS
;
        .MACRO  CNFDF$
        .MCALL  BLKDF$,MCB,MCBE,CBLK,RCT,HDR,DDB,DDBE
        .MCALL  UPAR,NCBD,NCB
BLKDF$
        .ENDM   CNFDF$
```

APPENDIX B

Example CNF Structure Construction

As an example of how to construct the CNF and NCB source files, consider a system consisting of the two diagnostics depicted in Figures 2-4 and 2-5. Figure B-1 depicts the combined configuration model; for simplification reasons, the partial Namlelist structure is not shown, but is indicated by a labeled pointer. The elements of each block which may be dynamically modified are shown in Figure B-2. The tree nodes are labeled by the first line in the node box and the contents of the node are given on the second line. All labels and contents are defined according to the following conventions:

1. The Header Block is designated by HDR and its content is the experiment short name.

2. DDBs are labeled as DDBnm where n is the diagnostic number and m is always zero. An abbreviated form of the diagnostic name is contained in the DDB node box.

3. MCBs are labeled as MCBnm, where n is the diagnostic number and m is the module number within the diagnostic. The MCB node box also contains the module type and unit number designation.

4. The partial Namelist pointers are labeled as NXnm corresponding to the associated node box label, where X is H, D or M depending upon the node type (HDR, DDB or MCB).

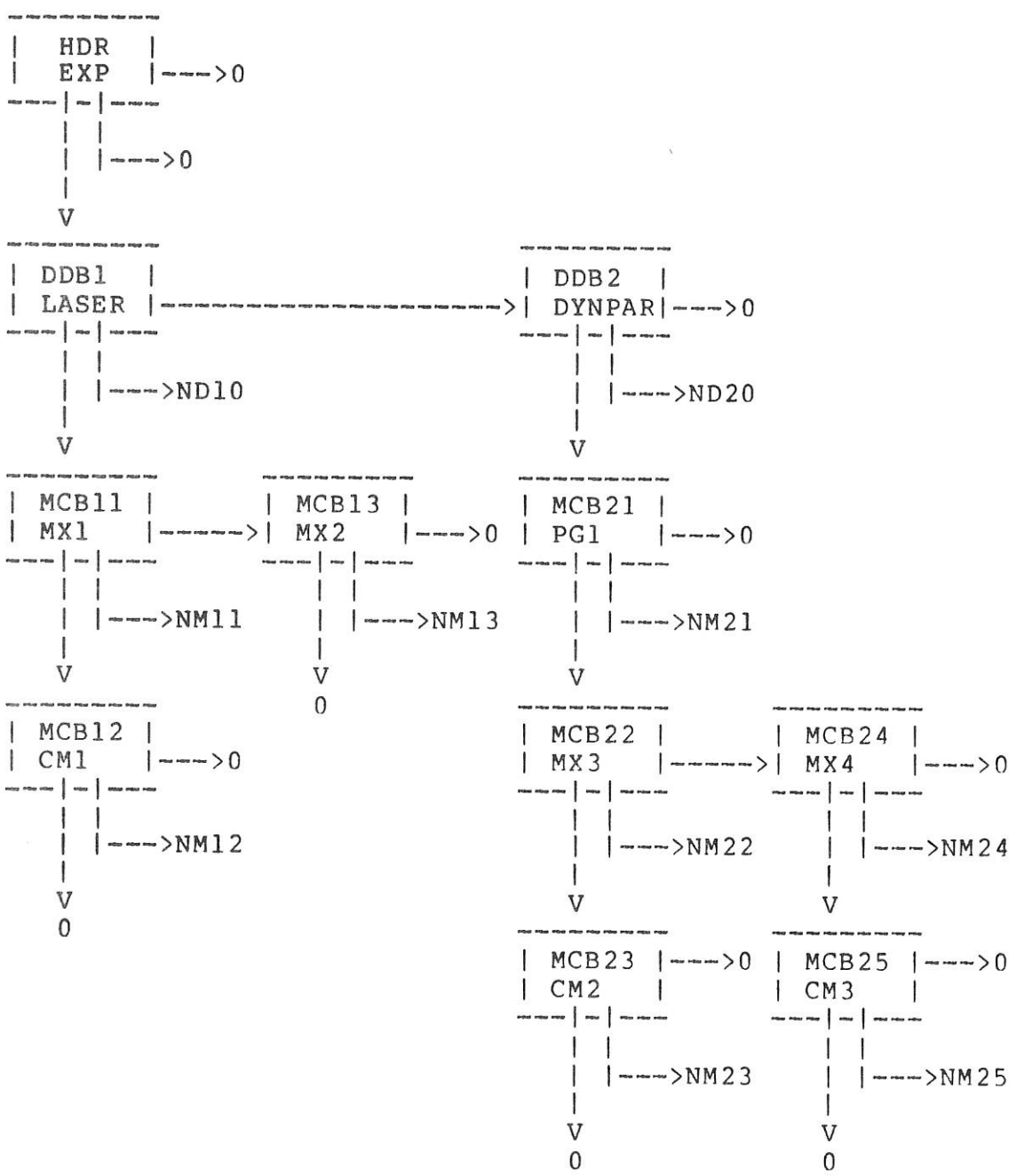


Figure B-1
Example Configuration Model

Block	Entry	Meaning
HDR		none
DDB1	ONLINE	indicates that the diagnostic is on-line or off-line. Initialized to off-line (0).
	FILTER	user parameter giving a filter factor as a real value. Initialized to 1.
MCB11	MODE	the MX operating mode
	ECHAN	the MX end channel
MCB12	DCT	number of data items to read and store in the data file.
MCB13	MODE	the MX operating mode.
	ECHAN	the MX end channel
	EXTMAP	the MX extender map
	FACTOR	user parameter giving the calibration factor. Initialized to 0.0031415.
DDB2	ONLINE	gives the diagnostic status as on-line or off-line. Initialized to off-line (0).
MCB21	PLSFRQ	specifies the duration and number of pulses to be generated by the PG as a vector of 32. real values. Initialized to all zero (generate no pulses).
MCB22	ECHAN	the MX end channel
MCB23	DCT	the number of data items to read and store in the data file.
MCB24	ECHAN	the MX end channel
MCB25	DCT	the number of data items to read and store in the data file.

Figure B-2
Modifiable Block Entries

In scanning the tree and specifying the macros which define the nodes, the following algorithm should be used:

1. Begin with the Header Block.

2. Follow the bottom (dependent) pointer to the next block and specify it.

3. Repeat Step 2 until the bottom pointer is zero.

4. Return up the tree via the bottom and then the side (independent) pointers until a non-zero side pointer is found; if none is found, then scanning is finished.

5. Follow the side pointer to the next block and specify it; then go to Step 2.

Returning to the example system and applying the above algorithm, the configuration file may be constructed as follows. The first step, as discussed in the section describing the macros, is to define the macros themselves. This is done with the following two statements:

```
.MCALL  CNFDF$
CNFDF$
```

The Header Block may now be defined with the statement:

```
HDR     1,DDB1,,<EXP>
```

which defines the Header Block as being one record (64 bytes) long, the next dependent block is the DDB labeled DDB1, and the experiment short name is "EXP". The Header Block has no specific NCBD's associated with it, thus the NCBD pointer is empty. Following the bottom pointer from the Header Block, the next block to be defined is DDB1; this is achieved with the following statements:

```
DDB1:   DDB     1,1,DDB2,MCB11,ND10
        DDBE    1,<LASER DIAGNOSTIC>,100,101,DC.DAQ
        UPAR    D.P01,U.FLT,1.0
```

in which the DDB length is given as one record, the diagnostic identification number is one, the next (independent) block is DDB2 and next dependent block is MCB11 and the block specific NCBD list is labeled ND11. The block extension macro designates the name of the diagnostic, specifies the owner UIC as [100,101] and defines the diagnostic as being for data acquisition. Note that the remaining parameters are not explicitly given, thus implying that the initial status is off-line. Furthermore, the first user param-

eter (FILTER) is defined as a REAL*4 variable and given the initial value of 1.

Continuing with the scanning algorithm, the bottom pointer of block DDB1 points to the block MCB11 which may be specified as follows:

```
MCB11:  MCB     1,1,MCB13,MCB12,NM11
        MCBE    <MX>,1,1,<EX>,<MC.HLD!MC.CAM>,,,,,,1200
        UPAR    M.P01,U.INT,0
        UPAR    M.P02,U.BYT,3
        UPAR    M.P02+1,U.ASC,<S>
```

The above macros define the first MCB as belonging to diagnostic 1, as having a partial Namelist beginning at location NM11 in the NCBD file, and as having a dependent module (MCB12) and independent module (MCB13). Furthermore, the module is a MUXADC with unit number 1 and module identifier 1. It is located in CAMAC crate 1 at station 8, the associated driver is in the EXACP task and no data are to be directly read (data are stored in the CAMMEM module defined by MCB12). The three UPAR macros define the MUXADC extender map (M.P01) as zero (i.e., there are no extender channels), defaults the end channel to 3 (M.P02) and sets the mode to "sequential scan". Note that the latter two parameters may be changed dynamically via the DLG task as variables "MODE" and "ECHAN" defined in the partial Namelist NM11.

The next module encountered by following the bottom pointer from the MCB11 block is MCB12. This is a CAMMEM CAMAC module with unit 1 and module indentification 2. The module CM1: is located in crate 1 station 9 and is to be read by the ACQ task following the post-trigger. The MCB is thus defined by the following macros:

```
MCB12:  MCB     1,1,,,NM12
CTL=MC.CAM!MC.GEN!MC.PST
        MCBE    <CM>,1,2,<EX>,<CTL>,U.INT,MF.MUX,1000,,,1220
        UPAR    M.P01,U.INT,1
        UPAR    M.P02,U.INT,0
        UPAR    M.P03,U.ASC,<E>
```

The device specific parameters M.P01, M.P02 and M.P03 define the core map (one 2K CAMMEM), the internal address from which data reading is to start and that data will be sent from another module via the external bus. Note that the number of data items to be read (1000) as defined in the MCBE macro may be dynamically changed via the DLG task as variable DCT defined in the partial Namelist NM12.

Returning to the scanning algorithm, the current node (MCB12) has no non-zero pointers. Referring to step 4 of the algorithm, the scanning is backed up the tree until a

node is found with a side (independent) pointer. Such a node is MCB11 which points to MCB13. This module may be defined with the following macros:

```
MCB13:  MCB     1,1,,,NM13
CTL=MC.CAM!MC.GEN!MC.PRE
        MCBE    <MX>,2,3,<EX>,<CTL>,U.INT,MF.MUX,24,,,1100
        UPAR    M.P01,U.INT,0
        UPAR    M.P02,U.BYT,7
        UPAR    M.P02+1,U.BYT,<R>
        UPAR    M.P05,U.FLT,3.1415E-2
```

Note that the user parameter FACTOR is assigned to location M.P05 in the user parameter area since the MX requires 8 bytes for device dependent parameters. Also, data are read directly from this module (in contrast to MX1:) and the first read request is initiated directly after receipt of the pre-trigger.

The node MCB13 has no non-zero pointers so that step 4 of the scanning algorithm is again relevant. Tracing back the tree, the next node containing a non-zero independent pointer is DDB1 which points to DDB2. The second diagnostic is defined with the macros:

```
DDB2:   DDB     1,2,,MCB21,ND20
        DDBE    1,<DYNAMIC PARAMETERS>,100,102,DC.DAQ
```

The second diagnostic belongs to the user with UIC [100,102]. The modules contained in diagnostic 2 are defined with the macros:

```
MCB21:  MCB     2,2,,MCB22,NM21
        MCBE    <PG>,1,1,<EX>,<MC.CAM!MC.HLD>,,,,,,2100
        UPAR    M.P01,<U.FLT*32.>
```

Note that the module must be held active during the shot (MC.HLD set) since other modules are externally connected to it and depend upon the output of PG1: for their proper functioning. Also, 32 real values for the specification of the pulse bursts are reserved by the UPAR macro. The block generated is two records long due to the number of device dependent parameters which would otherwise oveflow a single length block. Continuing with the tree scan, the remaing modules are defined by:

```
MCB22:  MCB     1,2,MCB24,MCB23,NM22
        MCBE    <MX>,3,2,<EX>,<MC.CAM!MC.HLD>,,,,,,2120
        UPAR    M.P01,U.INT,0
        UPAR    M.P02,U.BYT,7
        UPAR    M.P02+1,U.ASC,<S>
```

```
MCB23:  MCB     1,2,,,NM23
CTL=MC.CAM!MC.GEN!MC.PST
        MCBE    <CM>,2,3,<EX>,<CTL>,U.INT,MF.MUX,1400,,,2160
        UPAR    M.P01,U.INT,1
        UPAR    M.P02,U.INT,0
        UPAR    M.P03,U.ASC,<E>

MCB24:  MCB     1,2,,MCB25,NM24
        MCBE    <MX>,4,4,<EX>,<MC.CAM!MC.HLD>,,,,,,2300
        UPAR    M.P01,U.INT,0
        UPAR    M.P02,U.BYT,3
        UPAR    M.P02+1,U.ASC,<S>

MCB25:  MCB     1,2,,,NM25
CTL=MC.CAM!MC.GEN!MC.PST
        MCBE    <CM>,3,5,<EX>,<CTL>,U.INT,MF.MUX,1600,,,2340
        UPAR    M.P01,U.INT,1
        UPAR    M.P02,U.INT,0
        UPAR    M.P03,U.ASC,<E>
```

The above macros generate a model of two independent MUXADCs which feed data to one CAMMEM each and are commonly driven from one PG. All of the modules are located in CAMAC crate 2 starting at station number 4. As defined, a total of 12 analog channels are to be converted, the data stored in CM units and read after the post-trigger is received. Referring to the scanning algorithm, the tree is retraced according to step 4; no more nodes are found which contain non-zero independent pointers so that the trace ends at the Header node. It may thus be concluded that all nodes in the configuration model have been defined and that the next step in generating the model, the specification of the partial Namelists, may be begun.

Before beginning the construction of the Namelist source file for the example configuration, a few of the idiosyncracies of the file structure should be noted. The partial Namelists may generally be divided into two groups: privileged system-wide lists, one for each block type (Header Block, Diagnostic Descriptor Block and Module Control Block) and block-specific lists, one for each block. As described in the chapter on generating the configuration, the system-wide partial lists must begin at a defined record number in the NCB file: the first record must be the first element in the system-wide Header Block Namelist; the second record must be the first element in the system-wide DDB Namelist; the third record must be the first element in the system-wide MCB Namelist. If any of these lists is empty, a null element must be generated to fill the record; this is illustrated in the example below. The location of all other elements in the file is immaterial. A further point is that partial lists may themselves be made up of partial strings. This feature is particularly useful in referring to standard

offsets in particular blocks.

In constructing the NCB source file, the system-wide partial lists should be considerd first, starting with the Header Block. In the example, there is to be no system-wide Header Block list, therefore a null element must be generated. As in the CNF file, the first two statements must define the configuration block structure; the first statements in the file are as follows:

```
        .MCALL CNFDF$
        CNFDF$
HDR::   NCBD  NS.NDF
```

The null element for the system-wide Header Block list is done. The system-wide DDB list is to contain elements for modifying three entries in every DDB: the diagnostic identification (D.DID), the diagnostic user code (D.UOC) and the diagnostic user group code (D.UGC). Recalling the positional dependence of the start of the system-wide lists, only one element will be defined now, the definition of the remaining two will be deferred. The first element of the partial list is therefore:

```
DDB::   NCBD  NS.PRV,DBSW1,D$HP0,<DID>,D,D.DID,U.BYT
```

which allows the modification of the D.DID entry in the DCB by privileged users. This entry is a single byte entry (U.BYT length) and has an informational message in the User Message File at record D$HP0. The remainder of the list begins at DBSW1.

The first element of the MCB system-wide list must now be defined. This list is considerably longer than the corresponding DDB list and should contain entries which enable modification of the following elements: the module type (M.TYP), the module unit number (M.UNIT), the module physical address (M.ADR), and the control bits (MC.GEN, MC.PRE, MC.PST, MC.HLD). Since there are no positional records following this partial list, all elements of the MCB system-wide list may now be specified:

```
MCB::   NCBD  NS.PRV,MCSW1,M$HP1,<TYPE>,A,M.TYP,U.INT
MCSW1:  NCBD  NS.PRV,MCSW2,M$HP2,<UNIT>,I,M.UNIT,U.BYT
MCSW2:  NCBD  NS.PRV,MCSW3,M$HP9,<ADDR>,O,M.ADR,U.INT
MCSW3:  NCBD  NS.PRV,MCSW4,M$HC04,<GEN>,B,M.CTL,2,MC.GEN
MCSW4:  NCBD  NS.PRV,MCSW5,M$HC06,<PRESHT>,B,M.CTL,2,MC.PRE
MCSW5:  NCBD  NS.PRV,MCSW6,M$HC07,<PSTSHT),B,M.CTL,2,MC.PST
MCSW6:  NCBD  NS.PRV,,M$HC08,<HOLD>,B,M.CTL,2,MC.HLD
```

The remaining DDB system-wide NCBDs may now be definied as follows:

```
DBSW1:  NCBD  NS.PRV,DBSW2,D$HP4,<OWNER>,Y,D.UOC,U.BYT
DBSW2:  NCBD  NS.PRV,,D$HP4,<GROUP>,Y,D.UGC,U.BYT
```

These macros complete the definition of the system-wide partial lists.

As was previously mentioned, partial lists may consist of a list of partial lists. This is particularly useful in defining the device dependent parameters of modules which occur more than once in the configuration, in the example, the modules MX and CM. The MX has three dependent parameters, one of which (the end channel) is to be dynamically modifiable. Its macro definition is:

```
MXC:    NCBD  NS.NPR,,M$H121,<ENDCHA>,I,M.P02,U.BYT
```

The CM also has three device dependent parameters, none of which is to be modifiable. However, the number of data items to read and store in the data file should be modifiable for each CM. The macro definition is:

```
CMC:    NCBD  NS.NPR,,M$HC01,<ITEMS>,D,M.DCT,U.INT
```

A common partial list is also required for the DDBs as each diagnostic should be declarable as on-line or off-line by the diagnostic user (non-priviledged) as well as by the system manager (privilegded user). Had this entry been placed in the system-wide DDB partial list, only priviledged users could specify the on-line characteristic. Note that although this entry is specified as non-priviledged (NS.NPR) it may only be altered by the diagnostic owner and priviledged users. The DDB common partial list is given by the macro:

```
DDBC:   NCBD  NS.NPR,,D$HP5,<ONLINE>,B,D.STS,1,ST.ONL
```

This completes the definition of the common partial Namelists. It should be noted that all of the Namelist elements defined up to this point have had entries in the User Message file. Furthermore, all standard elements have informational messages in the User Message file which is generated in the GALE system generation process under the file name USRFMT.MSG and is located under UIC [1,2] on the system device. Chapter 3 contains information on generating additional messages for use with user specified NCBDs.

The only remaining NCBDs are those which are block specific. Some of the blocks have no specific entries so these may be assigned directly to the common partial lists or to an empty block. The first step is to generate an empty block, as follows:

```
DUM:    NCBD  NS.NPR
```

The block specific NCBDs will now be defined in the order in which their associated control blocks were defined above. The Header Block has no NCBD's associated with it. The first DDB is associated with the partial list labeled ND10. Here it is desired to give a filtering factor as input to a data processing program. The required NCBD is:

```
ND10::  NCBD  NS.NPR,DDBC,,<FILTER>,E,D.P01,U.FLT
```

In addition to the DDB specific parameter <FILTER>, the user may also set the diagnostic on-line or off-line via the DDB common partial list DDBC.

The next block to be dealt with is MCB11. Figure B-2 specifies that the user is to be allowed access to the two parameters MODE and ECHAN which are hardware specific parameters. The ECHAN entry is contained in the MX common partial list MXC so that it need be defined here; the partial list for MCB11 is therefore:

```
NM11::  NCBD  NS.NPR,MXC,M$H123,<MODE>,A,M.P02+1,U.BYT
```

The block MCB12 requires that the number of data items to be read (DCT) be modifiable. This entry is contained in the CM common list and is thus defined with the statement:

```
NM12==CMC
```

The control block MCB13 requires an entry for the input of the MX extender map the MUX mode and a calibration factor <FACTOR> as well as the MX common parameter. Its partial Namelist is thus given by the statements:

```
NM13::  NCBD  NS.NPR,NM131,,<FACTOR>,E,M.P05,U.FLT
NM131:  NCBD  NS.NPR,NM132,M$H122,<EXTMAP>,O,M.P01,U.INT
NM132:  NCBD  NS.NPR,MXC,M$H123,<MODE>,A,M.P02+1,U.BYT
```

In this case, two points should be noted. First, the diagnostic owner may enter the parameters FACTOR, EXTMAP and MODE as well as the MX common parameter since the block specific partial list is linked to the common partial list; and second, the FACTOR entry is initialized to a value of 0.0031415 in the definition of the MCB in the CNF file. The definition of the first diagnostic is now completed.

The second diagnostic is considerably less work since most of the modules contained in it are also in diagnostic 1 as a glance at Figures B-1 and B-2 reveals. The block DDB2 contains no block specific parameters as may be seen from its definition in the CNF file. The NCBD list may be quickly dispensed with by defining its partial list as the DDB common list with the statement:

```
ND20==DDBC
```

The next block encountered defines the PG module. The associated MCB defines a vector of 32 real values which define the Pulse-Frequency sequence to be produced by the generator. The required list is generated with the statement:

```
NM21::  NCBD  NS.NPR,,M$H112,<PLSFRQ>,E,M.P01,<U.FLT*32.>
```

The user may thus enter up to 16 pairs of real values. The following MX-CM pairs may be defined with already available partial lists since the user may only change the end-channel on the MX and the number of data items to be read from the CM. The partial lists are therefore defined with:

```
NM22==MXC
NM23==CMC
NM24==MXC
NM25==CMC
```

The second diagnostic is thus defined. As in any assembly source file, both the CNF and NCBD source files must be ended with a .END statement.

The source files created (Figures B-3 and B-4) may now be translated into a format acceptable to the ACQ and DLG tasks with the indirect command file CNFGEN.CMD which is on the GALE distribution disk.

```
        .MCALL CNFDF$
CNFDF$
        HDR   1,DDB1,,<EXP>
DDB1:   DDB   1,1,DDB2,MCB11,ND10
        DDBE  1,<LASER DIAGNOSTIC>,100,101,DC.DAQ
        UPAR  D.P01,U.FLT,1.0
MCB11:  MCB   1,1,MCB13,MCB12,NM11
        MCBE  <MX>,1,1,<EX>,<MC.HLD!MC.CAM>,,,,,,1200
        UPAR  M.P01,U.INT,0
        UPAR  M.P02,U.BYT,3
        UPAR  M.P02+1,U.ASC,<S>
MCB12:  MCB   1,1,,,NM12
CTL=MC.CAM!MC.GEN!MC.PST
        MCBE  <CM>,1,2,<EX>,<CTL>,U.INT,MF.MUX,1000,,,1220
        UPAR  M.P01,U.INT,1
        UPAR  M.P02,U.INT,0
        UPAR  M.P03,U.ASC,<E>
MCB13:  MCB   1,1,,,NM13
CTL=MC.CAM!MC.GEN!MC.PRE
        MCBE  <MX>,2,3,<EX>,<CTL>,U.INT,MF.MUX,24,,,1100
        UPAR  M.P01,U.INT,0
        UPAR  M.P02,U.BYT,7
        UPAR  M.P02+1,U.BYT,<R>
        UPAR  M.P05,U.FLT,3.1415E-2
DDB2:   DDB   1,2,,MCB21,ND20
        DDBE  1,<DYNAMIC PARAMETERS>,100,102,DC.DAQ
MCB21:  MCB   2,2,,MCB22,NM21
        MCBE  <PG>,1,1,<EX>,<MC.CAM!MC.HLD>,,,,,,2100
        UPAR  M.P01,<U.FLT*32.>
MCB22:  MCB   1,2,MCB24,MCB23,NM22
        MCBE  <MX>,3,2,<EX>,<MC.CAM!MC.HLD>,,,,,,2120
        UPAR  M.P01,U.INT,0
        UPAR  M.P02,U.BYT,7
        UPAR  M.P02+1,U.ASC,<S>
MCB23:  MCB   1,2,,,NM23
CTL=MC.CAM!MC.GEN!MC.PST
        MCBE  <CM>,2,3,<EX>,<CTL>,U.INT,MF.MUX,1400,,,2160
        UPAR  M.P01,U.INT,1
        UPAR  M.P02,U.INT,0
        UPAR  M.P03,U.ASC,<E>
MCB24:  MCB   1,2,,MCB25,NM24
        MCBE  <MX>,4,4,<EX>,<MC.CAM!MC.HLD>,,,,,,2300
        UPAR  M.P01,U.INT,0
        UPAR  M.P02,U.BYT,3
        UPAR  M.P02+1,U.ASC,<S>
MCB25:  MCB   1,2,,,NM25
CTL=MC.CAM!MC.GEN!MC.PST
        MCBE  <CM>,3,5,<EX>,<CTL>,U.INT,MF.MUX,1600,,,2340
        UPAR  M.P01,U.INT,1
        UPAR  M.P02,U.INT,0
        UPAR  M.P03,U.ASC,<E>
        .END
```

Figure B-3
Example CNF Source File

```
        .MCALL CNFDF$
CNFDF$
;
;       SYSTEM-WIDE PARTIAL LISTS
;
HDR::   NCBD  NS.NDF
DDB::   NCBD  NS.PRV,DBSW1,D$HP0,<DID>,D,D.DID,U.BYT
MCB::   NCBD  NS.PRV,MCSW1,M$HP1,<TYPE>,A,M.TYP,U.INT
MCSW1:  NCBD  NS.PRV,MCSW2,M$HP2,<UNIT>,I,M.UNIT,U.BYT
MCSW2:  NCBD  NS.PRV,MCSW3,M$HP9,<ADDR>,O,M.ADR,U.INT
MCSW3:  NCBD  NS.PRV,MCSW4,M$HC04,<GEN>,B,M.CTL,2,MC.GEN
MCSW4:  NCBD  NS.PRV,MCSW5,M$HC06,<PRESHT>,B,M.CTL,2,MC.PRE
MCSW5:  NCBD  NS.PRV,MCSW6,M$HC07,<PSTSHT),B,M.CTL,2,MC.PST
MCSW6:  NCBD  NS.PRV,,M$HC08,<HOLD>,B,M.CTL,2,MC.HLD
DBSW1:  NCBD  NS.PRV,DBSW2,D$HP4,<OWNER>,Y,D.UOC,U.BYT
DBSW2:  NCBD  NS.PRV,,D$HP4,<GROUP>,Y,D.UGC,U.BYT
;
;       MX, CM AND DDB COMMON PARTIAL LISTS
;
MXC:    NCBD  NS.NPR,,M$H121,<ENDCHA>,I,M.P02,U.BYT
CMC:    NCBD  NS.NPR,,M$HC01,<ITEMS>,D,M.DCT,U.INT
DDBC:   NCBD  NS.NPR,,D$HP5,<ONLINE>,B,D.STS,1,ST.ONL
DUM:    NCBD  NS.NPR
;
;       BLOCK SPECIFIC PARTIAL LISTS
;
ND10::  NCBD  NS.NPR,DDBC,,<FILTER>,E,D.P01,U.FLT
NM11::  NCBD  NS.NPR,MXC,M$H123,<MODE>,A,M.P02+1,U.BYT
NM12==CMC
NM13::  NCBD  NS.NPR,NM131,,<FACTOR>,E,M.P05,U.FLT
NM131:  NCBD  NS.NPR,NM132,M$H122,<EXTMAP>,O,M.P01,U.INT
NM132:  NCBD  NS.NPR,MXC,M$H123,<MODE>,A,M.P02+1,U.BYT
ND20==DDBC
NM21::  NCBD  NS.NPR,,M$H112,<PLSFRQ>,E,M.P01,<U.FLT*32.>
NM22==MXC
NM23==CMC
NM24==MXC
NM25==CMC
        .END
```

Figure B-4
Example NCBD Source File

APPENDIX C

User Message File

The following is a listing of the User message file which is generated as USRFMT.MSG by the GALE system generation.

Label	Message
H$HP0	SHOT NUMBER OF NEXT DATA FILE TO BE WRITTEN
H$HPH1	TIME OF LAST SHOT HOUR MINUTE SECOND RESPECTIVELY THESE FIELDS ARE SET BY THE ACQ TASK
H$HPH2	DATA OF LAST SHOT DAY MONTH YEAR RESPECTIVELY THESE FIELDS ARE SET BY THE ACQ TASK
D$HP0	DIAGNOSTIC ID CODE IN RANGE 1 TO 255 (10)
D$HP1	BLOCK TYPE CODE: 1 - FOR HEADER 2 - FOR DDB 3 - FOR MCB
D$HP2	DIAGNOSTIC MESSAGE STRING LENGTH. THE MAX VALUE IS 21
D$HP3	USER MESSAGE STRING OF THE DIAGNOSTIC. THE VECTOR SIZE IS 21. CHARACTERS
D$HP4	UIC OF DIAGNOSTIC OWNER
D$HP5	BIT SET MEANS: DIAGNOSTIC IS ONLINE
D$HP11	BIT SET MEANS: DATA ACQUISITION DIAGNOSTIC
D$HP12	BIT SET MEANS: MONITORING AND CONTROL DIAGNOSTIC
D$HP13	USER PARAMETER SECTION

M$HP0 DIAGNOSTIC ID CODE
THE VALUE MUST MATCH THE CONTENTS OF DID IN THE DDB

M$HP1 2 CHARACTER ASCII MODULE TYPE CODE

M$HP2 PHYSICAL UNIT NUMBER OF THE MODULE

M$HP3 MODULE ID CODE IN DIAGNOSTIC

M$HP4 2 CHARACTER ASCII PARTIAL NAME OF THE ACP CONTAINING THE DRIVER FOR THE MODULE.
THIS ENTRY IS ZERO FOR BUILT-IN DRIVERS.

M$HP5 BIT SET MEANS: MODULE GENERATES INTERRUPT

M$HP6 BIT SET MEANS: MODULE IS A CAMAC MODULE

M$HP7 INTERRUPT VECTOR NUMBER COMPUTED BY DIVIDING ADDRESS BY 4 FOR NON-CAMAC INTERRUPTING MODULES; OTHERWISE ZERO.

M$HP8 INTERRUPT PRIORITY FOR NON-CAMAC INTERRUPTING MODULES; OTHERWISE ZERO.

M$HP9 DEVICE REGISTER OR CAMAC ADDRESS

M$HP111 LENGTH OF DEVICE DEPENDENT PARAMETER BUFFER IN BYTES

M$HP112 REAL VECTOR DEFINING THE PULSE SEQUENCE LIST
ONE BURST IS GIVEN THROUGH A PAIR OF REALS WITH THE FOLLOWING MEANING: THE FIRST VALUE GIVES THE DURATION (IN SECONDS), THE SECOND SETS THE NUMBER OF PULSES IN THAT BURST.
NOTE!
FOR TS THE MAX. NUMBER OF BURSTS IS 7
FOR PG THE MAX. NUMBER OF BURSTS IS 16

M$H121 MUX END CHANNEL NUMBER

M$H122 EXTENDER BIT MAP:
EACH BIT IS ASSOCIATED WITH THE CORRESPONDING MUX CHANNEL. A BIT SET MEANS: THE CORRESPONDING CHANNEL IS CONNECTED TO AN EXTENDER.

M$H123 MUX MODE:
R - RANDOM READ
S - SEQUENTIAL SCAN
T - SEQUENTIAL TRIGGERED

M$H131 CORE MAP BIT PATTERN DESCRIBING THE PHYSICAL CONFIGURATION OF THE LOGICAL CAMMEM

M$H132 START ADDRESS OF DATA TRANSFER

M$H133 ACCESS INDICATOR:
I - INTERNAL
E - EXTERNAL

M$HC01 NUMBER OF DATA ITEMS DESIRED

M$HC02 DATA FORMAT DESCRIPTOR:
0 - NON-STANDARD
1 - LOGICAL DATA
2 - INTEGER DATA
3 - REAL DATA
4 - MUX DATA

M$HC03 BIT SET MEANS: CONTROL MODULE

M$HC04 BIT SET MEANS: DATA GENERATION MODULE

M$HC05 LENGTH IN BYTES OF DATA ITEM

M$HC06 BIT SET MEANS: READ DATA BEFORE SHOT

M$HC07 BIT SET MEANS: READ DATA AFTER SHOT

M$HC08 BIT SET MEANS: DO NOT UNLOAD MODULE DURING DATA ACQUISITION

GALE System
Programmer's Handbook

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