MAX-PLANCK-INSTITUT FÜR PLASMAPHYSIK

GARCHING BEI MÜNCHEN

DIOS I/O Operations

Klaus Engelhardt
Robert Lathe
Erich Müller

IPP R/26 October 1977

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

K. Engelhardt
R. Lathe
E. Mueller

Abstract

This manual provides the information necessary to enable the reader to create and execute tasks using the DIOS I/O facilities.

CONTENTS

CONTENTS

CONTENTS

PREFACE

0.1 Manual Objectives and Reader Assumptions

The goal of this manual is to provide the necessary information to enable the reader to create and execute tasks which use the DIOS I/O facilities. The reader is assumed to be familiar with the RSX I/O system. This implies familiarity with the following documents:

1. RSX-11M I/O Procedures
2. RSX-11M Device Drivers
3. RSX-11M Executive Services
4. RSX-11M Operators Procedures

Further, the reader must be able to write and assemble MACRO-11 programs, and to build tasks with TKB.

Users who intend to generate the DIOS system must also be familiar with the manual "RSX-11M System Generation".

0.2 Structure of the Document

Chapter 1 gives an introduction to the purposes, features, and basic concepts of the DIOS system.

Chapter 2 describes the details of writing MACRO-11 programs to perform the basic DIOS operations.

Chapter 3 describes how to build DIOS user tasks with TKB.

Chapter 4 describes the steps needed to run DIOS user tasks.

Appendix A lists DIOS-related error messages and codes.

Appendix D describes the organization of DIOS-supported CAMAC systems.

Appendix E describes the procedures for generating a DIOS system.

Appendix F describes the procedure for updating driver ACP's.

0.3 Related Documents

DIOS Device Handlers

This document contains a description and functional specification of each standard module driver distributed with the DIOS system. Programmers should refer to this manual when writing programs which perform I/O to such modules.

Guide to Writing a DIOS I/O Driver

This document gives general specifications of a DIOS I/O driver. System programmers who intend to write drivers for nonstandard module types should refer to this document.

Guide to Writing an ACP

This document describes the RSX-11M I/O interface to Ancilliary Control Processors (ACP), upon which the DIOS system is based. Within the context of DIOS applications, it is of interest mainly as background information and for error diagnosis.

0.4 The GALE System

DIOS is made to be compatible with the Generalized Acquisition System for Laboratory Experiments, developed concurrently by the PDE at the IPP in Garching. The GALE system is described in the following documents.

Introduction to the GALE System

GALE Terminal User's Guide

GALE Programmer's Guide

0.5 Message Output Processor

The Message Output Processor (MO) is a general facility for printing formatted messages on the user terminal and the console log device. It is used by DIOS programs and user tasks, and should be included with the active DIOS system. Its generation and use are described in the GALE System Programmer's Handbook and GALE Programmers's Handbook respectively. MO is also distributed with the DIOS system.

CHAPTER 1

Introduction to DIOS

1.1 Purposes

The Dynamic I/O System (DIOS) was created to allow user tasks under RSX-11M to control experimental hardware without requiring that the controlling programs be permanently resident within the system. Instead, device control structures and drivers are dynamically loaded into core as needed and removed when hardware control is no longer desired. Since experimental hardware tends to be modified often during development, the system was designed to allow new drivers to be included and existing ones to be modified in a standardized and efficient process. Finally, the adherence to most RSX I/O conventions provides for substantial preservation of system integrity once the device drivers have been completely debugged.

1.1.1 Features

Features of the system include:

1. Support of two types of CAMAC system, either one or more Borer single-crate systems or a Schlumberger ICP-11 multi-crate system.

2. Availability of a set of standard module drivers for a wide range of Data Acquisition and Control needs.

3. Support of the standard RSX driver functions of I/O Cancellation, Device Timeout, and Powerfail Recovery.

4. Full capability of handling interrupts, including correct dispatching of module LAM's from all stations within the CAMAC system.

1.1.2 Restrictions

Several restrictions must be taken into account in evaluating the System's applicability to a given situation:

1. DIOS is compatible only with a mapped RSX-11M system.

2. Modules using more than one interrupt vector are not standard and require special programming, which may detract from system security.

1.2 Basic Concepts

DIOS is implemented as an extension of the RSX I/O system. Basic DIOS operations are carried out with QIO directives submitted by user tasks: in addition, the system may be activated and deactivated by MOU and DMO commands from the terminal.

1.2.1 Modules

Devices which are to be controlled using the DIOS system are referred to as modules. Each module is identified by a module type code and a unit number. The module type code consists of two letters; it is shared by all modules with identical hardware function. The unit number distinguishes modules of the same type; for example, "MX0" and "MX1" denote two PDE/IPP Multiplex-ADC modules.

Two types of interface concepts are generally supported: these are the PDP-11 Unibus and the CAMAC system.

Modules on the Unibus have one or more device registers, in a block beginning at a definite register address. They may or may not generate interrupts; if so, they may only use a single vector.

CAMAC modules occupy one or more stations of a CAMAC crate connected to the PDP-11 via a controller. The specifications of the crate controller systems and the CAMAC modules are discussed in Appendix D.

CHAPTER 1

Introduction to DIOS

1.1 Purposes

The Dynamic I/O System (DIOS) was created to allow user tasks under RSX-11M to control experimental hardware without requiring that the controlling programs be permanently resident within the system. Instead, device control structures and drivers are dynamically loaded into core as needed and removed when hardware control is no longer desired. Since experimental hardware tends to be modified often during development, the system was designed to allow new drivers to be included and existing ones to be modified in a standardized and efficient process. Finally, the adherence to most RSX I/O conventions provides for substantial preservation of system integrity once the device drivers have been completely debugged.

1.1.1 Features

Features of the system include:

1. Support of two types of CAMAC system, either one or more Borer single-crate systems or a Schlumberger ICP-11 multi-crate system.

2. Availability of a set of standard module drivers for a wide range of Data Acquisition and Control needs.

3. Support of the standard RSX driver functions of I/O Cancellation, Device Timeout, and Powerfail Recovery.

4. Full capability of handling interrupts, including correct dispatching of module LAM's from all stations within the CAMAC system.

1.1.2 Restrictions

Several restrictions must be taken into account in evaluating the System's applicability to a given situation:

1. DIOS is compatible only with a mapped RSX-11M system.

2. Modules using more than one interrupt vector are not standard and require special programming, which may detract from system security.

1.2 Basic Concepts

DIOS is implemented as an extension of the RSX I/O system. Basic DIOS operations are carried out with QIO directives submitted by user tasks: in addition, the system may be activated and deactivated by MOU and DMO commands from the terminal.

1.2.1 Modules

Devices which are to be controlled using the DIOS system are referred to as modules. Each module is identified by a module type code and a unit number. The module type code consists of two letters; it is shared by all modules with identical hardware function. The unit number distinguishes modules of the same type; for example, "MX0" and "MX1" denote two PDE/IPP Multiplex-ADC modules.

Two types of interface concepts are generally supported: these are the PDP-11 Unibus and the CAMAC system.

Modules on the Unibus have one or more device registers, in a block beginning at a definite register address. They may or may not generate interrupts; if so, they may only use a single vector.

CAMAC modules occupy one or more stations of a CAMAC crate connected to the PDP-11 via a controller. The specifications of the crate controller systems and the CAMAC modules are discussed in Appendix D.

1.2.2 Module Drivers

For each module type, there is an associated driver program which converts I/O requests from a user task into the specific Unibus or CAMAC commands recognized by the given type of module.

Drivers for standard modules are distributed with the DIOS system: these are listed and described in "DIOS Device Handlers".

If the user wishes to support a module not included in the standard DIOS repertoire, he must write his own driver according to the specifications given in "Guide to Writing a DIOS I/O Driver".

1.2.3 Driver ACP's

To be used with the DIOS system, a module driver must be included within a larger task called the Driver ACP, which may contain other module drivers as well. An ACP is a task associated with a given I/O device which is activated whenever an I/O request for that device is received, and which supervises processing of the request. The operating principles of an ACP are discussed in "Guide to Writing an ACP".

The Driver ACP is normally installed in the system, but is not active. When an I/O request for a module whose driver is within the ACP arrives, the ACP is activated and begins processing the I/O request by calling the module driver as a subroutine. The driver ACP remains active until all I/O requests for drivers within it are finished; then it waits for more requests or exits if no more I/O is expected.

There may be several driver ACP's within the system: each one is identified by a 2-character driver ACP ID. The task name under which the ACP is installed is formed from the driver ACP ID by appending four periods. For example, the driver ACP "W7" is installed as task W7.... .

1.2.4 Loading Modules

In order to use a DIOS module, the user task must first request that the module be loaded. This operation involves

1. Logically defining the module within the DIOS system, by allocating and initializing data structures describing the module. If the same module (as determined by its module type and unit) was already loaded, the previously allocated data structures are used.

2. Activating the driver ACP specified in the load request.

3. Assigning a specified LUN to the module.

4. Accessing a file on the module LUN. This operation is required for RSX to pass I/O requests to an ACP.

1.2.5 Module I/O Operations

The main purpose of the DIOS system is to perform I/O operations on modules. This can be done only when the module is loaded.

I/O operations are executed by issuing QIO directives to the LUN assigned to the module. DIOS I/O conventions are substantially the same as those for RSX. These conventions are discussed in "RSX I/O Operations", "RSX I/O Drivers" and "RSX Executive" (QIO-associated directives).

The possible I/O functions and their application are determined by the module driver. The manual "DIOS Device Handlers" gives this information for the distributed DIOS drivers.

1.2.6 Unloading Modules

When a task has finished using a module, a request to unload the module may be issued. This operation involves

1. Deaccessing the file on the module LUN and deassigning the LUN, freeing it for use with other devices.

2. Removing the module structures from the system if no other tasks are using the module.

1.2.7 The Loader Device -- DA0:

The heart of the DIOS system is the Loader Device. It is incorporated within the RSX system as device unit DA0:, which is a "pseudodevice" in that it does not correspond to any real physical device on the Unibus. Like other RSX devices, DA0: has device data structures and a device driver; these reside in a partition lying within the RSX executive region.

As far as user tasks are concerned, the only function of the loader device is to process requests to load modules, submitted via QIO directives. To issue load requests the task must first assign a LUN to DA0:, and then access a file on this LUN.

In addition, the loader is responsible for activating and deactivating the DIOS system. DIOS is activated by "mounting" DA0: with the MOU command; it may be deactivated with a DMO command.

Finally, the loader device driver dispatches the "emergency" functions supported by standard RSX drivers: I/O cancellation, device timeout, and powerfail recovery. The loader device determines when such an event occurs and causes the module driver to take appropriate action.

1.2.8 The Loader ACP

The Loader ACP is associated with device DA0:; it is considered a part of the loader device and performs the following functions:

1. Process requests to load modules.
2. Process requests to access a file on the loader.
3. Process terminal commands to mount and dismount DA0:.
4. Remove data structures for unloaded modules.

It is installed in the system with task name DA...., and is activated when DA0: is mounted.

1.3 Using the DIOS System

Using DIOS involves

1. Writing assembler programs to perform DIOS operations
2. Assembling the programs
3. Building tasks from the assembled programs
4. Setting up and activating DIOS
5. Running the tasks.

1.3.1 Prerequisites

1.3.1.1 Hardware -

1. PDP-11 with memory management and a minimum hardware configuration to support an RSX-11M V 3.0 mapped system.
2. The hardware modules and interfaces to be used.

1.3.1.2 Software -

1. RSX-11M V 3.0 (mapped) operating system for the PDP-11. The detailed prerequisites for the host RSX system are given in Appendix E.
2. DIOS subsystem generated for the given installation as described in Appendix E.

CHAPTER 2

Writing Assembler Programs to use DIOS

2.0 Introduction -- General Notes

This chapter describes the tools available to the assembler programmer for carrying out the basic DIOS operations described in section 1.2. These operations are summarized briefly here, in the order in which they would normally be used.

1. Assign LUN to Loader
2. Access Loader
3. Load Module
4. Perform Module I/O
5. Unload Module
6. Deaccess Loader

With the exception of assigning the loader LUN, all operations are performed by issuing a QIO directive for the corresponding function. Requests to load and unload a module may also be submitted by using simplified DIOS macro calls.

2.0.1 Macro Calls

The MCB and MCBE macro calls must be used in the given order. In all other cases, the first line is the preferred version; subsequent lines are alternative macro calls.

Full argument lists are given for LOAD and UNLOAD.

The full argument lists for MCB and MCBE are described in "GALE System Programmer's Handbook".

The full argument lists for ALUN$-, QIO$-, and QIOW$- are given in "RSX Executive".

All system directives (ALUN$-, QIO$-, and QIOW$-) are given in the -$S form; any other form (-$C or -$ with DIR$) may be used if the form of the arguments is changed accordingly (see "RSX Executive"). LOAD and UNLOAD generate equivalent QIOW$S macro calls.

2.0.2 Arguments

The first line contains a description of the general type of argument:

-$S system directives, LOAD, and UNLOAD treat arguments as sources of MOV instructions. The argument is an immediate constant equal to the described quantity, or a register or word location containing the quantity.

MCB, MCBE, and the -$C and -$ forms of system directives require the arguments to be MACRO-11 symbolic expressions equal to the described quantity.

All arguments of LOAD and UNLOAD are described in general.

Arguments of MCB and MCBE are described in full in "GALE System Programmer's Handbook".

Arguments of QIO$-, QIOW$-, and ALUN$- are described in general in "RSX Executive". Only those with particular significance to DIOS are described here.

2.0.3 Directive Errors

ALUN$-, QIO$-, QIOW$-, LOAD, and UNLOAD submit system directives to RSX, as described in "RSX Executive".

If the directive is accepted by RSX, control is returned with the carry bit clear and a positive success code in the low byte of the DSW. Unless otherwise indicated, this code is always equal to IS.SUC.

If the directive is rejected, control is returned with carry set. The low byte of the DSW contains a negative error code. The possible codes resulting from the described application of the directives are listed under each operation. Only those codes resulting from specific violations of DIOS conventions are listed. Others rarely occur, and if they do, are described under the specific directive in "RSX Executive".

2.0.4 I/O Errors

I/O Errors are returned in the first word of the I/O status block if one was specified in the QIO$-, QIOW$-, LOAD, or UNLOAD.

If the directive was rejected, the first word is undefined.

If a QIO$- directive was accepted but the I/O operation is not yet complete, the first word is zero.

If the I/O operation is complete and successful, the low byte is positive. Unless otherwise indicated, it contains the code IS.SUC. The high byte depends on the I/O operation but is usually zero.

When the I/O operation is complete, a negative error code returned in the first byte of the status block indicates an error or other unusual condition occurred. A group code uniquely identifying the error is returned in the high byte; the interpretation of the error and group codes is discussed in Appendix A.

In the description of the DIOS operation, the symbolic value of the error code is given, along with its standard message and a brief description of the circumstances which could cause the error. Only those errors caused by specific violations of DIOS conventions are listed. Other errors rarely occur, and are described in "RSX I/O Operations" and "RSX I/O Drivers". Errors from module I/O operations are listed in the description of the module driver.

2.0.5 Second Status Word

The second word of the I/O status block may return additional information.

If the directive is rejected it is undefined.

If a QIO$- was accepted but is not complete, it is zero.

When the I/O operation is complete (successful or not) it contains the listed status information. If the I/O operation was to a module, its contents are given in the driver description.

2.0.6 Synchronization of I/O Operations

All DIOS operations submitted with a QIO$- directive are asynchronous, that is, the I/O operation may not be complete when control is returned to the issuing task. This includes load, unload and access loader operations requested with a QIO$-. A QIOW$- directive submitted without specifying an event flag is also asynchronous.

While an asynchronous operation is not complete, the event flag, if specified, is clear and the status block set to zero.

When the I/O operation is completed, the event flag is set and status information returned to the I/O status block. The AST routine specified in the QIO$- or QIOW$- is entered immediately. When the AST is finished, the task resumes normal operation.

A task may wait for I/O to complete by issuing a WAITFOR directive on the event flag specified in the QIO$-.

LOAD, UNLOAD, and a QIOW$- specifying an event flag are always synchronous: that is, the task does not regain control until the I/O operation is completed. AST's may not be specified with the LOAD or UNLOAD macros; if given with a QIOW$-, the AST is executed before normal operation is resumed.

The following operations should always use synchronous forms:

-- Access and Deaccess Loader
-- Load and Unload Module
-- Virtually Attach and Detach Module (IO.VAT, IO.VDT)
-- Cancel I/O (IO.KIL)

With these operations, only two cases arise in which the task could continue executing before the operation is complete:

1. If the loader or driver ACP has a lower task priority than the issuing task; this is not usually the case.

2. If the loader or driver ACP is checkpointed when the request is received; the user task could continue running while the ACP is read into core, but the time gained is insignificant.

2.1 Assigning LUN to Loader

Requests for the loader to perform DIOS operations are submitted via QIO directives to device DA0:; a LUN must be assigned to DA0: before such requests can be accepted.

Macro Call:

```
ALUN$S  #$LDR,#"DA,#0
```

Arguments: Immediate or word location

DA	Two-letter ASCII name of loader device.
0	Unit number of loader device.
$LDR	Global symbol equated to loader LUN.

Directive Errors:

IE.IDU ILLEGAL DEVICE OR UNIT
The loader device DA0 is not present in the RSX configuration. Make sure the DIOS system has been properly installed (Chapter 4 or Appendix E).

IE.ILU ILLEGAL LUN
$LDR was not equated to a valid LUN for the task.

Notes:

1. The symbol $LDR is not required if the LOAD and UNLOAD macros are not used (see below). If used, it may be defined as in the following example:

   ```
   $LDR == 3
   ```

   This example uses LUN 3 for the loader device.

2. Use of the ALUN$ directive is optional. If it is not used, the LUN must be assigned with the ASG option when the task is built. The symbol $LDR may also be defined at this time with the GBLDEF option. See Chapter 3 for details.

2.2 Accessing Loader

The effect of this operation is to declare that a file is open on the LUN of the loader device. This allows requests to load modules to be processed. Attempts to reassign the loader LUN are blocked while the file is open.

Macro Calls:

```
QIOW$S   #IO.ACW,#$LDR,[flg],,[sts],[ast]
QIO$S    #IO.ACW,#$LDR,[flg],,[sts],[ast]
```

Arguments: Immediate or word location

```
IO.ACW   I/O function code "Access File for Write"
$LDR     Global symbol equated to loader LUN.
```

Other arguments are standard for a QIO directive.

Directive Errors:

IE.ULN UNASSIGNED LUN
The LUN was not assigned to DA0: (or any other device).
Possibly $LDR was incorrectly defined.

I/O Errors:

IE.ALN FILE ALREADY ACCESSED ON LUN
The loader was already accessed when the request was issued.

IE.PRV PRIVILEGE VIOLATION
DA0: has not been mounted (see Chapter 4).

Second Status Word: zero

Notes:

1. QIOW$- is preferred.

2.3 Loading a Module

A task wishing to perform DIOS I/O operations must first load the intended module. From the task's point of view, it simply specifies the module type, unit number, and interface characteristics; a LUN, also specified, is then assigned to the module and a file is accessed on the LUN. Subsequent QIO directives for this LUN are then translated into the appropriate module action.

Internally, DIOS takes the following actions, depending on the status of the module:

* If the module was not previously loaded, data structures are allocated and initialized with the interface characteristics submitted in the load request. The driver ACP containing the module driver is activated. The data structures are then labeled with the module type and unit number, and the module is considered loaded.

* At this point, or if the module was already loaded, the specified LUN is assigned to the module data structures. In addition, a file is accessed on the module LUN.

2.3.1 Module Control Block Structure

The vehicle for passing the module characteristics to DIOS is the Module Control Block (MCB), which is allocated and initialized within the task requesting the load. A pointer to the MCB is passed to DIOS in the load request. The structure of the MCB is illustrated in the following diagram:

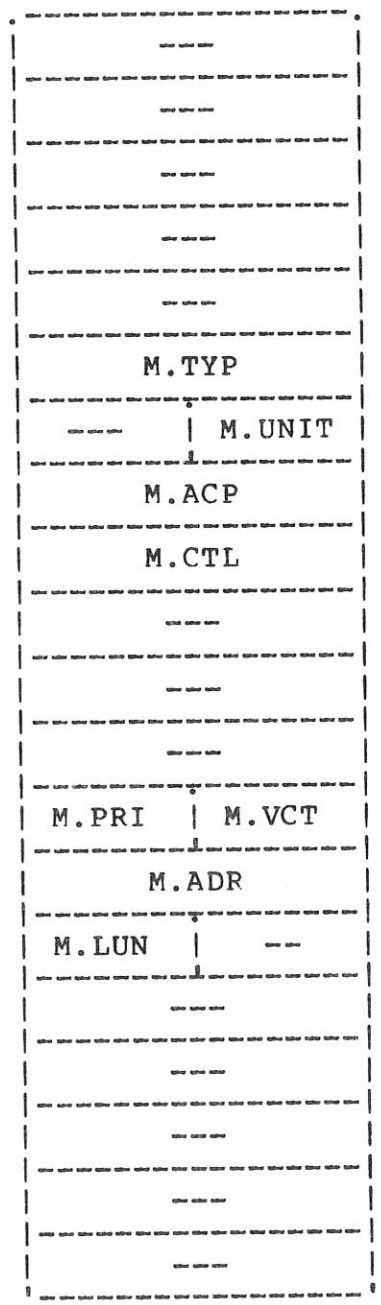

Fig 2.1 Module Control Block Structure

2.3.2 Module Control Block Contents

Name	Size in Bytes	Contents
--	12	Unused
M.TYP	2	Module Type Code 2 alphabetic ASCII characters

Identifies the module type being loaded. M.TYP and M.UNIT are first checked to determine if the module is already loaded. If not, M.TYP is used to locate the proper module driver within the driver ACP; it is then recorded to identify the data structures allocated for the loaded module.

M.UNIT	1	Module Unit Number Unsigned integer 0. - 255.

Distinguishes the module from others of the same type. M.UNIT and M.TYP are checked to determine if this module is already loaded. If not, M.UNIT is recorded to identify the data structures allocated for the module.

--	1	Unused
M.ACP	2	Driver ACP ID 2 ASCII characters

Identifies the driver ACP containing the module driver. The ACP must be installed with task name xx...., where xx denotes the contents of M.ACP.

M.CTL	2	Control Bits 16-bit logical mask

Contains the following flags describing the module:

MC.INT	If set, module requires interrupt or LAM service.
MC.CAM	If set, module is in CAMAC system.
MC.SUB	If set, module is a sub-module [1]

All other bits are unused.

--	6	Unused

---

[1] The sub-module concept may be used by non-standard drivers for modules which are interfaced through a multiplexer module. In this case, the controller is a normal DIOS module with a CAMAC or Unibus address in M.ADR, and MC.SUB = 0. Each submodule 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 marked as an invalid Unibus or CAMAC address.

M.VCT 1 Interrupt Vector Location
Unsigned Integer, 16(8) - 377 (8)

Interrupt vector address divided by 4.
The vector address must be a multiple of 4 and lie between 70(8) and the highest vector address allowed (see Appendix E).
If the module does not require interrupt service, MC.INT is not set and M.VCT need not be specified.

M.PRI 1 Interrupt Priority
200(8) - 340(8) (bits 5-7 only)

Contains the interrupt priority of the module, encoded in bits 5-7 (upper 3 bits). The encoded priority must lie between 4 and 7. The lower 5 bits (0 - 4) must be zero.
If the module does not require interrupt servicing, M.PRI need not be specified.

M.ADR 2 Module Address
16 - bit logical

For a non-CAMAC module, contains the Unibus address (160000 to 177556) of the lowest module device register (usually a command/status register (CSR)).

If MC.CAM is set, contains the CAMAC address of the module:
Low byte: Station Number
High byte: Crate Number
(See Appendix D for details of the CAMAC system).

If MC.SUB is set, the contents of this field are determined by the drivers of the module and controller.

-- 1 Unused

M.LUN 1 Module LUN, returned from load
Unsigned integer, 1 - 255.

This field must be reserved before issuing the load request. After the load request, this field contains the LUN assigned to the module. If the request was unsuccessful, the contents are unspecified.

-- 12 Unused

M.DDP 0 End of module control block

2.3.3 Using Macros to Define the MCB

The MCB may be allocated and initialized at assembly time with the two macros described in this section. They are primarily used with the GALE system, which has an extended version of the MCB in which the DIOS MCB is embedded; those arguments which initialize the unused portions of the MCB are omitted from the present description.

The MCB is defined with the following assembler statements:

```
label:
        MCB     1,1,0,0,0
        MCBE    <type>,unit,,<acp>,ctl,,,,vct,pri,adr
```

Arguments: Symbolic assembler expressions

type	Module type code (enclosed in angle brackets)
unit	Module unit number
acp	Driver ACP name (enclosed in angle brackets)
ctl	Control bits in M.CTL.
vct	Interrupt vector address (not divided by 4!)
pri	Interrupt priority (in bits 0 - 2 !)
adr	Module address

label Used to refer to MCB in load request.

NOTE

The MCBE macro already converts the priority and the vector address to the proper bit positions for the MCB, so the programmer does not have to do this.

2.3.4 Executing the Load Request

Macro Calls:

```
LOAD    lun,mcb,[sts],[flg]
QIOW$S  #IO.LOD,#$LDR,[flg],,[sts],[ast],<mcb,#M.DDP,lun>
QIO$S   #IO.LOD,#$LDR,[flg],,[sts],[ast],<mcb,#M.DDP,lun>
```

Arguments: Immediate or word location

IO.LOD	I/O function code "Load Module"
$LDR	LUN of the loader device
flg	(optional) Event flag to be set when load is complete. If the LOAD form is used and flg is not specified, event flag 32. is assumed by default.
sts	(optional) I/O status block address.
lun	LUN to be assigned to module.
mcb	MCB address.
M.DDP	length of MCB.

Directive Errors:

IE.ILU ILLEGAL LUN
The loader LUN specified is not legal for the task. Make sure that $LDR was properly defined.

I/O Errors:

IE.NLN NO FILE ACCESSED ON LUN
The loader was not accessed.

IE.ILU ILLEGAL LUN
The LUN specified for the module is not legal for the task. Make sure enough units are allocated when the task is built.

IE.ALN FILE ALREADY ACCESSED ON LUN
A file was already accessed on the specified module LUN. Make sure the specified LUN is correct and is not being used by another device.

IE.BAD BAD PARAMETERS
The specified length of the MCB is too small; it should be M.DDP bytes.

IE.IDU ILLEGAL DEVICE OR UNIT
The MCB contains illegal or incorrect specifications:

-- The module address coincides with that of another loaded module or RSX device.
-- The CAMAC or Unibus address has illegal range or format.
-- The specified interrupt vector address is out of range.
-- The specified interrupt vector is being used by another module or RSX device.

IE.OFL DEVICE OFFLINE
For a non-CAMAC module, the specified Unibus address is incorrect, or the module is not connected to the Unibus.
For a CAMAC module, the specified crate is not online, or a nonexistent crate was specified (see Appendix D). Even if the module is not present, IE.OFL is not returned as long as the crate is online.

IE.ACP DRIVER NOT INSTALLED
The driver ACP specified is not installed, has an incorrect name (not xx....), or the ACP ID is incorrect.

IE.DRV DRIVER NOT FOUND IN ACP
No driver for the specified module type was included in the ACP. Make sure the module type and ACP-ID are correct, or rebuild the ACP to include the desired driver.

IE.UPN INSUFFICIENT DYNAMIC STORAGE
Module data structures could not be allocated due to lack of dynamic memory space. Increase the size of the pool with SET /POOL, or remove unneeded tasks.

Notes:

1. The same error code, IE.ULN, may be returned if either the loader LUN or module LUN specified is incorrect. The programmer should check whether a directive error or an I/O error occurred in order to determine which LUN is illegal.

2. If a module of the same type and unit number was already loaded, DIOS does not process the other MCB parameters. Instead, the LUN is assigned to the module previously loaded, and a file is accessed on the LUN. The module has the characteristics defined when it was first loaded. Whether or not this occurred is of no concern to the I/O task -- I/O processing continues in exactly the same way as if the module had been newly loaded.

3. LOAD or QIOW$- is the preferred form.

2.4 Performing Module I/O Operations

This section describes the use of QIO$- and QIOW$- directives in performing I/O operations on loaded modules. In general, the module driver determines which I/O functions are valid for a given module type. The detailed implications of driver I/O functions are given in the corresponding chapter of "DIOS Device Handlers".

In addition to the driver functions, there are four functions performed directly by DIOS which have identical effects for all modules.

2.4.1 Standard I/O Functions

The four standard I/O operations are

```
QIO$S   #IO.VAT,lun,...,<#0,#4> ;Attach Module
QIO$S   #IO.VDT,lun,...,<#0,#4> ;Detach Module
QIO$S   #IO.KIL,lun,...         ;Cancel Pending I/O
QIO$S   #IO.UNL,lun,...         ;Unload Module
```

With the exception of IO.UNL, they are described in the following section. IO.UNL is described in section 2.5 .

2.4.1.1 IO.VAT -- Attach Module (Virtual) -

IO.VAT is a substitute for the standard RSX function IO.ATT. On successful completion, the module is attached to the issuing task. I/O requests submitted from other tasks are queued, but are not executed until the module is detached with IO.VDT.

IO.ATT may not be used, for RSX does not allow devices which have an ACP to be attached.

Macro Call:

```
QIOW$S  #IO.VAT,lun,[flg],,[sts],[ast],<#0,#4>
QIO$S   #IO.VAT,lun,[flg],,[sts],[ast],<#0,#4>
```

Arguments: Immediate or word location

```
IO.VAT  I/O function code "Attach Module (Virtual)"
lun     LUN of module
0       Dummy buffer address (required)
4       Dummy buffer length (required)
```

Directive Errors: Standard

I/O Errors:

IE.DAA DEVICE ALREADY ATTACHED
The module has already been attached by this task.

Second Status Word: zero

Notes:

1. QIOW$- is preferred.

2.4.1.2 IO.VDT -- Detach Module (Virtual) -

IO.VDT is a substitute for the RSX function IO.DET. On successful completion, the module is detached from the issuing task. Execution of I/O requests from other tasks, which was blocked while the module was attached, is now started.

IO.DET may not be used because RSX does not allow devices having an ACP to be attached and detached.

Macro Calls:

```
QIOW$S  #IO.VDT,lun,[flg],,[sts],[ast],<#0,#4>
QIO$S   #IO.VDT,lun,[flg],,[sts],[ast],<#0,#4>
```

Arguments: Immediate or word location

```
IO.VDT  I/O Function Code "Detach Module (Virtual)"
lun     LUN of module
0       Dummy Buffer Address (required)
4       Dummy Buffer Length (required)
```

Directive Errors: Standard

I/O Errors:

IE.DNA DEVICE NOT ATTACHED
The module was not attached to the issuing task.

Second Status Word: zero

Notes:

1. QIOW$- is preferred.

2.4.1.3 IO.KIL -- Abort Pending I/O Requests -

IO.KIL allows a task to abort all unfinished I/O requests it has submitted to the module.

Outstanding requests may be active or queued. Since the driver can only process one request at a time, only one can be active. The rest are queued for later processing.

On IO.KIL, all queued requests from the issuing task are simply removed from the queue for the module and completed, returning the error code IE.ABO to the status block. The event flag is set, but the AST routine, if specified, is not entered.

If the active request was from the issuing task, the module driver is called to abort the request. It returns the module to an inactive state and only then dismisses the active request, returning error code IE.ABO in the status block. In some cases, where the request is sure to finish in a relatively short time, the driver may simply allow it to expire naturally, in which case the error code IE.ABO is not returned. The event flag is always set; whether the AST is called, depends on the driver.

Macro Call:

```
QIOW$S    IO.KIL,lun,[flg],,[sts],[ast]
QIO$S     IO.KIL,lun,[flg],,[sts],[ast]
```

Arguments: Immediate or word location

```
IO.KIL  I/O function code "Cancel Outstanding I/O"
lun     LUN of module
```

Directive Errors: Standard

I/O Errors: Standard

Notes:

1. QIOW$- is preferred.

2.4.2 Driver-Specific I/O Functions

All other I/O requests for loaded modules are processed by the module driver itself. If a distributed DIOS driver is used, the format and action of each I/O request is fully discussed in the driver description contained in "DIOS I/O Drivers". If a user-written driver is being used, the legal I/O functions, formats and actions may be obtained from the driver writer.

Macro Call:

```
QIO$S   fcn,lun,[flg],,[sts],[ast],<p1,p2,p3,p4,p5,p6>
QIOW$S  fcn,lun,[flg],,[sts],[ast],<p1,p2,p3,p4,p5,p6>
```

Arguments: Immediate or word location

fcn	I/O function to execute
lun	LUN of module
flg	(optional) Event flag set when I/O complete
sts	(optional) I/O status block
ast	(optional) AST routine called when I/O complete
p1...p6	(optional) Parameter list

Directive Errors:

IE.ULN UN-ASSIGNED LUN
The module was never loaded.

I/O Errors:

IE.NLN NO FILE ACCESSED ON LUN
No module was loaded under this LUN.

IE.IFC ILLEGAL FUNCTION CODE
The module driver does not recognize the function code specified.

IE.BAD BAD PARAMETERS
One or more items in the parameter list is invalid. See driver for details.

IE.OFL DEVICE OFFLINE
The module does not exist or did not respond to any commands from the driver.

IE.IDS INCONSISTENT WITH DEVICE STATE
Usually this means the module must be initialized to perform this function. See driver for details.

IE.FHE FATAL HARDWARE ERROR
Some hardware malfunction was detected by the driver. See driver for details.

IE.EOV END OF VOLUME DETECTED
An attempt was made to read or write more data than the module can hold.

IE.DAO DATA OVERRUN
On reading a record, data at the end of the record was lost because the input data buffer was too small.

IE.RBG ILLEGAL BUFFER SIZE
On a read or write operation, the buffer size specified was not an exact multiple of the number of bytes in a single data item for the module.

IE.DNR DEVICE NOT READY
The module failed to perform the desired function within a certain period of time, depending on the module and driver.

IE.ABO REQUEST ABORTED
The task aborted outstanding I/O on the module by issuing an IO.KIL request.

IE.PWF REQUEST ABORTED DUE TO POWERFAIL
A powerfail occurred and the driver could not recover.

Second Status Word:

The contents depend on the module driver.

On read/write functions, this word (almost) always contains a byte count specifying the number of bytes actually read or written. This number is less than or equal to the number of bytes requested in the QIO$-. It may be less due to one of the following circumstances:

1. More data was requested than contained in the next record.

2. An End of Volume was detected before the request was satisfied.

3. A hardware error was detected before the request was satisfied.

Consult the driver for the specific condition. In any case this value should be used when manipulating data read from the module.

On non-read/write functions, other status information may be returned in this word, depending on the driver.

Notes:

1. Most of the error codes listed above are semi-standard in that their specific meanings depend on the driver. In particular, the hardware status of the module after the error may vary. Exceptions are IE.NLN and IE.IFC, which are the same for all modules.

2.4.3 Semi-Standard Functions

The I/O functions described in this section are semi-standard in that if they are used by a module driver, they should perform the actions described below. These con-

ventions hold for any driver written for use with the GALE data acquistion task.

The following functions are described:

```
QIO$C   IO.RVB,lun,...,<buf,len>    ;Read Virtual Block
QIO$C   IO.WVB,lun,...,<buf,len>    ;Write Virtual Block
QIO$C   IO.TER,lun,...,<0,4>        ;Terminate Module
```

2.4.3.1 IO.RVB -- Read Virtual Block -

This function is generally used to read a requested number of data bytes from the module into a buffer within the user task. The effects of repeated attempts to read data depend on the type of module; two main categories may be distinguished.

* Modules delivering a limited amount of data:

All of the data from such modules is typically read out in one request. If the buffer specified is too small to hold all the data, the error code IE.DAO is returned. The data may usually be read repeatedly.

* Modules delivering a large amount of data:

The IO.RVB function reads only the requested amount of data. If more data are available, a subsequent IO.RVB will transfer data starting from the last item read. If more data are requested than are available, the error code IE.EOV is returned. In order for the module to be read again, it must be initialized with IO.INI.

Macro Calls:

```
QIO$S    IO.RVB,lun,[flg],,[sts],[ast],<buf,lbuf>
QIOW$S   IO.RVB,lun,[flg],,[sts],[ast],<buf,lbuf>
```

Arguments: Immediate or word location

```
IO.RVB  I/O function code "Read Virtual Block"
lun     module LUN
buf     Address of buffer to hold data read from module
lbuf    Number of bytes of data to read
```

Directive Errors: Standard

I/O Errors:

IE.EOV END OF VOLUME DETECTED
IE.DAO DATA OVERRUN
IE.RBG ILLEGAL BUFFER SIZE

Second Status Word:

Number of bytes actually read.

2.4.3.2 IO.WVB -- Write Virtual Block -

This function is generally used to write a requested amount of data to a module. Successive applications of IO.WVB produce different effects, depending on the module; two main cases may be distinguished:

* Modules accepting a limited amount of data:

All the data accepted by the module must be written in a single request. If IO.WVB is repeated, the previous data are overwritten.

* Modules accepting a large amount of data:

The IO.WVB writes exactly the requested amount of data. If the module can accept more data, successive IO.WVB functions may be performed. If the module cannot accept more data, the error code IE.EOV is returned. In order to write more data to the module, an IO.INI must be issued.

Macro Calls:

```
QIO$S    IO.WVB,lun,[flg],,[sts],[ast],<buf,lbuf>
QIOW$S   IO.WVB,lun,[flg],,[sts],[ast],<buf,lbuf>
```

Arguments: Immediate or word location

```
IO.WVB  I/O function code "Write Virtual Block"
lun     Module LUN
buf     Buffer containing data to be written
lbuf    Amount of data in bytes
```

Directive Errors: Standard

I/O Errors:

IE.EOV END OF VOLUME DETECTED
IE.RBG ILLEGAL BUFFER SIZE

Second Status Word:

Number of bytes actually written.

2.4.3.3 IO.INI -- Initialize Module -

This function sets the module to an initial state specified in the parameter buffer given in the QIO. If no parameters are required, the null buffer may be specified. The choice of whether parameters are used, and what they mean, is left to the driver.

Macro Calls:

```
QIO$S    IO.INI,lun,[flg],,[sts],[ast],<prm,lprm>
QIOW$S   IO.INI,lun,[flg],,[sts],[ast],<prm,lprm>
```

Arguments:

```
IO.INI  I/O function code "Initialize Module"
lun     Module LUN
prm     Address of parameter buffer
lprm    Size of parameter buffer
```

Directive Errors: Standard

I/O Errors: Depend on driver

Second Status Word: Depends on driver

2.4.3.4 IO.TER -- Terminate Module -

This function is typically used to return a module to a quiescent state. Its application is most important for those modules carrying on some computer-independent process which may need to be stopped under certain conditions.

Macro Calls:

```
QIO$S    IO.TER,lun,[flg],,[sts],[ast],<4,0>
QIOW$S   IO.TER,lun,[flg],,[sts],[ast],<4,0>
```

Arguments: Immediate or word location

```
IO.TER  Function Code "Terminate Module"
lun     Module LUN
<0,4>   Dummy buffer address and length, required
        because IO.TER is a subfunction of IO.WVB.
```

Directive Errors: Standard

I/O Errors: Depend on driver

Second Status Word: Depends on driver

2.5 Unloading a Module

When a task no longer needs to perform I/O on a module, it may issue a request to unload the module. This operation causes the following actions:

1. Deaccess the file on the module LUN
2. Deassign the module LUN, preventing further QIO directives from being accepted.
3. If the module is not loaded by another task or under another LUN of the issuing task, the module data structures are removed from the system. If the driver ACP is not needed to service other loaded modules, the ACP is deactivated.

Macro Calls:

```
UNLOAD  lun,[sts],[flg]
QIOW$S  #IO.UNL,lun,[flg],,[sts],[ast]
QIO$S   #IO.UNL,lun,[flg],,[sts],[ast]
```

Arguments: Immediate or word location

```
IO.UNL  I/O function code "Unload Module".
lun     Module LUN.
flg     (optional) Event flag set when unload complete.  If
        UNLOAD is used, omitting flg causes event flag 32.
        to be used by default.
sts     (optional) Address of I/O status block.
```

Directive Errors: Standard

I/O Errors:

```
IE.NLN  NO FILE ACCESSED ON LUN
        No module was loaded under this LUN.
```

Second Status Word: Zero

Notes:

1. UNLOAD or QIOW$- is preferred.
2. Unloading the module is optional. If omitted, it is performed automatically when the task exits as part of the I/O rundown sequence.

2.6 Deaccessing Loader

This request declares that the issuing task no longer needs to load any modules. In detail, the following actions are performed:

1. Deaccess file on loader LUN. Further load requests from this task will not be honored, and the LUN may be reassigned to another device with ALUN$-.

2. The loader ACP will exit if no modules are left loaded, no other task or LUN has a file accessed on the loader, and DA0: has been dismounted.

Macro Calls:

```
QIOW$S  #IO.DAC,#$LDR,[flg],,[sts],[ast]
QIO$S   #IO.DAC,#$LDR,[flg],,[sts],[ast]
```

Arguments: Immediate or word location

```
IO.DAC  I/O function code "Deaccess File on LUN"
$LDR    Global symbol for loader LUN
```

Directive Errors: Standard

I/O Errors:

```
IE.NLN  NO FILE ACCESSED ON LUN
        The loader LUN was not accessed.
```

Second Status Word: zero

Notes:

1. The QIOW$ form of the request is preferred.

2. The deaccess LUN request may be omitted. If so, the request will automatically be performed when the user task exits as a part of I/O rundown.

2.7 Defining Macros and Symbols

All of the macros mentioned in this chapter are contained in the System Macro Library (RSXMAC.SML). Before being used, they must be defined using the .MCALL assembler directive, as follows:

To use RSX directives:

```
.MCALL  QIO$S,QIOW$S,ALUN$S, ...
```

To use LOAD and UNLOAD:

```
.MCALL  LOAD,UNLOAD
```

To use the MCB allocation macros:

```
.MCALL  MCB,MCBE
```

To define DIOS Symbols:

```
.MCALL  MCBDF$,DIODF$
```

The symbols for the DIOS function and error codes are defined as follows:

```
DIODF$  ;DEFINE DIOS FUNCTION AND ERROR CODES
.PSECT  <previous program section>
```

The MCB offset symbols are defined as follows:

```
MCBDF$  ;DEFINE MCB OFFSET SYMBOLS
.PSECT  <previous program section>
```

NOTE

> The .PSECT is required to restore the previous program section, which was changed by the preceding macro.

MCBDF$ and DIODF$ may be omitted; in this case any references to the symbols will be resolved when the task is built by including global definitions from the system library modules MCBSYM, DIOSYM, and QIOSYM.

If MCB's are being defined with MCB and MCBE, MCBDF$ must be invoked to define the offsets.

CHAPTER 3

Building Tasks to Use DIOS

Once the programs discussed in chapter 2 are assembled, they may be built into a DIOS user task using the Task Builder. This section discusses DIOS-related conventions which apply to building the task.

3.1 Allocating I/O units

Enough units must be allocated for the task for all devices which the task intends to use simultaneously. This includes all RSX device units, the DIOS loader device, and all modules to be loaded at the same time. If error reporting is done with MO (see Appendix A), a LUN for MO0: must also be included.

The allocation is done by specifying the number of LUN's which may be assigned with the UNITS option when the task is built, as follows:

```
  ...
ENTER OPTIONS:
TKB>UNITS=n
  ...
```

where n is the total number of units; note that n is also the highest LUN which may be assigned.

3.2 Assigning the Loader LUN

Instead of using the ALUN$ directive, the programmer may assign the loader LUN when the task is built using the ASG option. The global symbol $LDR may also be defined with the GBLDEF option. This method is usually preferred, as it eliminates the ALUN$- directive from the program.

In the following example, LUN 3 is assigned to the loader.

```
  ...
ENTER OPTIONS:
TKB>GBLDEF=$LDR:3
TKB>ASG=DA0:3
  ...
```

3.3 Assigning the Message Output LUN

If error reporting is being done with message output, a LUN must be reserved and assigned to MO0:, as in the following example.

```
ENTER OPTIONS:
TKB>ASG=MO0:1
```

3.4 Example of a Task Build Command File

This section contains an example of an indirect command file submitted to TKB to build a typical DIOS user task.

```
; TASK BUILD COMMAND FILE FOR IOTASK:
;
; IOTASK USES THE FOLLOWING LUNS:
;
;       1       MO0: (MESSAGE OUTPUT)
;       2       NOT USED
;       3       DA0: (LOADER)
;       4       NOT USED
;       5       TI: (TERMINAL)
;       6       TI: (TERMINAL)
;       7-10.   MODULES
;
DK2:IOTASK,IOTASK=DK2:IOTASK
/
UNITS=10          ;ALLOCATE LUN'S
GBLDEF=$LDR:3     ;DEFINE SYMBOL FOR LOADER LUN
ASG=DA0:3         ;ASSIGN LOADER LUN
ASG=MO0:1         ;ASSIGN MESSAGE OUTPUT LUN
ASG=TI:5:6        ;ASSIGN TERMINAL LUN
//
```

CHAPTER 4

Running DIOS User Tasks

This chapter explains the steps necessary to prepare the RSX/DIOS system for running tasks using DIOS, running the tasks, and deactivating the DIOS system when it is no longer needed.

Each command typed by the user is described; the error messages which can appear on the terminal are then listed and their probable cause is given. Only those error messages due to specific violations of DIOS conventions are listed; others which may occur are explained in the RSX Operator's Procedures manual.

4.1 Installing the DIOS System

If the RSX system has been saved with DIOS installed, or if DIOS has already been installed since RSX was last booted, this step is not necessary. Otherwise, the command

```
@[1,2]INSDIOS.CMD
```

must be executed. The command file loads the MO driver, installs the MO ACP, and mounts MO:. Then it installs the DIOS system-resident routines into DAPAR, runs DIOLNK to link the loader device DA: into the RSX device list, installs the loader and driver ACP's, and mounts DA0:. The details of installing DIOS are described in Appendix E. Normally, a command to execute INSDIOS is included in the system startup file to perform this automatically whenever RSX is booted.

Error Messages:

The following error messages may be received when executing INSDIOS.CMD. The errors are usually caused if INSDIOS is accidentally executed with DIOS already installed. Preceding each group of error messages is the command causing them.

```
>INS [1,54]LOA
INS -- TASK NOT IN SYSTEM
```
The LOA command processor is not available. It is required to support loadable drivers, like MO.

```
>LOA MO:
LOA -- OPEN FAILURE ON FILE MODRV.STB
```
The loadable driver for MO: was not generated.

```
>INS [1,54]MOACP
INS -- TASK NOT IN SYSTEM
```
The message output ACP was not generated.

```
>INS [1,54]DADRV
INS -- PARTITION NOT IN SYSTEM
```
Common partition DAPAR is not in the system.

```
INS -- PARTITION NOT COMMON
```
Partition DAPAR has been converted to a task partition to prevent its being overwritten.

```
>RUN [1,54]DIOLNK
 DEVICE DA: ALREADY LINKED TO SYSTEM
```
This message is returned from DIOLNK if it is run when DIOS has already been installed.

```
>INS [1,54]LDRACP
>INS [1,54]xxACP
INS -- FILE NOT FOUND
```
The loader or driver ACP task images are not available.

4.2 Activating the DIOS System

If DIOS was installed but not activated, the following commands must be executed:

```
MOU MO:
MOU DA:
```

The first command mounts the message output processor, needed by the loader ACP. The second command activates the DIOS system; in detail, MOU DA: does the following:

1. Allocates temporary DIOS data structures.
2. Activates DIOS processes, in particular the periodic scanning of loaded modules for expired timeouts.
3. Initializes the CAMAC system. The software and hardware actions taken are described in detail in Appendix D.
4. Activates the Loader ACP, which then waits for I/O requests from user tasks.

After the MOU DA: command, DEV will show device DA0 as mounted.

NOTE

The last two commands are performed by INSDIOS and thus must not be repeated if INSDIOS is executed.

Error Messages:

MOU -- CANNOT MOUNT DEVICE
The MOU command processor has not been replaced by the DIOS version, DIOMOU.TSK.

MOU -- DEVICE NOT IN SYSTEM
The specified device (DA0: or MO0:) is not in the system; DIOS has not been completely installed.

MOU -- ACP NOT INSTALLED
The corresponding ACP (DA.... or MO.... for DA0: and MO0:, respectively) has not been installed.

MOU -- ALREADY MOUNTED
The device (DA0: or MO0:) has already been mounted.

4.3 Running the User Task

The task may be run using any valid RSX task initiation procedure, usually the RUN command.

Error Messages:

INS -- ILLEGAL DEVICE DA0:
DIOS has not been installed.

4.4 Deactivating the DIOS System

DIOS may be deactivated by the commands

```
DMO DA0:
DMO MO0:
```

The first command does the following:

1. Determines if any DIOS user tasks are still using DIOS. This is true if

   -- any modules are still loaded
   -- any task still has a file accessed on DA0:.

   If so, DA0: is marked for dismount, and the remaining steps are performed only when these conditions no longer hold.

2. Terminates all active DIOS processes. In particular, this includes the scanning of loaded modules for expired timeouts.

3. Resets the CAMAC system; the detailed actions involved are discussed in Appendix D.

4. Deallocates temporary DIOS data structures.

5. Deactivates the Loader ACP.

6. Causes the following message to be printed on the console log:

   *** DA0: -- DISMOUNT COMPLETE

The results of a DEV command issued after DMO DA0: are the following:

* If modules are still loaded or the loader is accessed, DA0: will appear as "MARKED FOR DISMOUNT".

* When DIOS has been deactivated, DA0: will appear as not mounted.

The DMO MO0: command is used to deactivate the message output processor; it should not be issued if message output is still desired by other tasks.

Error Messages:

DMO -- ALREADY MARKED FOR DISMOUNT
The DMO command has already been issued, but the dismount is not complete because modules are still loaded and the loader accessed.

DMO -- NOT MOUNTED
DA0: or MO0: was not mounted when the DMO command was issued.

DMO -- DEVICE NOT IN SYSTEM
A DMO command was given when DIOS was not installed or MO: not loaded.

APPENDIX A

RSX/DIOS Error Messages and Codes

A variety of run-time error conditions can arise when using DIOS I/O, the RSX I/O and file systems, or other facilities. This appendix describes the classification and reporting of such errors.

A.1 Classification of Errors

The errors discussed in this appendix occur as the result of incorrectly applying system directives or macro calls, or from hardware failures on devices during QIO operations. The user is notified of the occurrence of an error by a variety of means. In each case, however, the error condition is identified by means of an error code returned in a status region associated with the operation. This section describes the general form of the error code, and lists each possible cause of errors, giving the detailed error code conventions applying to each case.

A.1.1 Error Codes and Groups

Errors are classified into two main groups -- Group 0 and Group 1 -- depending on the cause of the error. Within a group, each type of error is identified by a unique error code between -1. and -128.

Programs refer to the error codes by symbols of the general form IE.xxx, where xxx denotes a mnemonic for the actual error condition. The only exceptions are the GCML error symbols GE.IOR, GE.OPR, GE.BIF, and GE.MDE.

NOTE

Two different errors from different groups may have the same error code value (with different symbols). For example, the codes IE.BAD (group 0) and IE.UPN (group 1) are both equal to 377(8).

A.1.2 Directive Errors

If a system directive (ALUNS, QIOS, etc.) is issued with improper arguments or under otherwise improper conditions, the directive is rejected. Immediately after the macro call, the following holds:

1. The carry bit is set, indicating an error occurred.
2. The error code is returned in the low byte of the Directive Status Word (DSW) at location $DSW.

Directive errors are classified under Group 1.

If the carry bit is not set, the directive was accepted: the low byte of the DSW is positive.

A.1.3 I/O Errors from RSX Devices

If a QIO directive to an RSX device was accepted, and the driver or RSX I/O system detected an error, the I/O request is completed and the error code returned in the first byte of the I/O status block (if specified in the QIO).

RSX I/O errors are classified under Group 0.

If the code is nonnegative, the I/O was successful or is not finished.

A.1.4 I/O Errors from DIOS

If a QIO directive to a DIOS module or the loader was accepted, and the module driver, DIOS, or the RSX I/O system detected an error, the I/O request is completed and an error code returned in the first word of the I/O status block (if

specified in the QIO). This word has the following format:

Byte 0 Error Code
Byte 1 Group Code (0 = Group 0; 1 = Group 1)

The above applies also to the LOAD and UNLOAD macros.

If Byte 0 is nonnegative, the I/O was successful or is not finished.

A.1.5 Errors from the RSX File System

If a File I/O macro was processed and the RSX file system determined that an error occurred, the operation is terminated and the error code is returned in the File Descriptor Block at locations F.ERR and F.ERR+1:

F.ERR Error Code
F.ERR+1 If 0, an RSX Device I/O error occurred (Group 0)
If negative, a directive error occurred (Group 1)

A.1.6 Errors from the GCML Macros

If an error occurred while processing a GCML$, RCML$ or CCML$ macro, the error code is returned in the GCML control block at byte location G.ERR.

GCML errors are classified under Group 1.

A.2 Reporting Errors with Message Output

This section describes the use of the Message Output processor to report errors as they occur by typing an error message on the user terminal or console log device.

A.2.1 Error Messages

Associated with the Message Output processor is a System Message File (usually named [1,2]MOTFMT.MSG) which contains a standard message for each of the errors discussed in this appendix. The file is blocked into 64-byte records; the message format strings are referred to by record number. Errors are associated with messages as follows:

Group n:
  record # = -(error code) + n*128.

A.2.2 Printing Error Messages

Before any error messages may be printed, the following steps must be taken:

* Make sure MO has been generated and is mounted.

* Assign a LUN to MO0:. This may be done either with the ALUN$- directive or with the ASG option when the task is built.

* Open a file on the message output LUN, by including the following macro call within the user program:

```
MOOP$S  lun,[flg],[sts]
```

Arguments: Immediate or word location

lun	LUN assigned to MO0:
flg	Event flag to be set on completion
sts	I/O status block

A WAITFOR directive for the specified event flag should be issued to make sure the open is complete.

MO is now ready to report error messages from this task. When an error occurs, the record number of the message must be calculated as described above. Then the message may be printed with the following macro call:

```
MOUT$S  lun,[flg],[sts],,,,rec,dst
```

Arguments: Immediate or word location

lun	LUN assigned to MO0:
flg	Event flag set when the message has been printed
sts	I/O status block
rec	Record number of message
dst	Terminal on which message is to be printed:

dst > 0: Message appears on CL: and TI:.
dst = 0: Message appears on TI: only.
dst < 0: Message appears on CL: only.

A WAITFOR directive on the specified event flag must be issued to make sure the output is finished.

A complete description of the Message Output processor and its use is given in "GALE Programmer's Handbook".

A.3 List of Error Codes and Messages

Errors from Group 0:

Code	Value (octal)	Record (dec.)	Message
IE.BAD	377	1.	BAD PARAMETERS
IE.IFC	376	2.	INVALID FUNCTION CODE
IE.DNR	375	3.	DEVICE NOT READY
IE.VER	374	4.	PARITY ERROR ON DEVICE
IE.ONP	373	5.	HARDWARE OPTION NOT PRESENT
IE.SPC	372	6.	ILLEGAL USER BUFFER
IE.DNA	371	7.	DEVICE NOT ATTACHED
IE.DAA	370	8.	DEVICE ALREADY ATTACHED
IE.DUN	367	9.	DEVICE NOT ATTACHABLE
IE.EOF	366	10.	END OF FILE DETECTED
IE.EOV	365	11.	END OF VOLUME DETECTED
IE.WLK	364	12.	WRITE ATTEMPTED TO LOCKED UNIT
IE.DAO	363	13.	DATA OVERRUN
IE.SRE	362	14.	SEND/RECEIVE ERROR
IE.ABO	361	15.	REQUEST TERMINATED
IE.PRI	360	16.	PRIVILEGE VIOLATON
IE.RSU	357	17.	SHARABLE RESOURCE IN USE
IE.OVR	356	18.	ILLEGAL OVERLAY REQUEST
IE.BYT	355	19.	ODD BYTE COUNT (OR VIRTUAL ADDRESS)
IE.BLK	354	20.	LOGICAL BLOCK NUMBER TOO LARGE
IE.MOD	353	21.	INVALID UDC MODULE
IE.CON	352	22.	UDC CONNECT ERROR
IE.NOD	351	23.	CALLER'S NODES EXHAUSTED
IE.DFU	350	24.	DEVICE FULL
IE.IFU	347	25.	INDEX FILE FULL
IE.NSF	346	26.	NO SUCH FILE
IE.LCK	345	27.	LOCKED FROM READ/WRITE ACCESS
IE.HFU	344	28.	FILE HEADER FULL
IE.WAC	343	29.	ACCESSED FOR WRITE
IE.CKS	342	30.	FILE HEADER CHECKSUM FAILURE
IE.WAT	341	31.	ATTRIBUTE CONTROL LIST FORMAT ERROR
IE.RER	340	32.	FILE PROCESSOR DEVICE READ ERROR
IE.WER	337	33.	FILE PROCESSOR DEVICE WRITE ERROR
IE.ALN	336	34.	FILE ALREADY ACCESSED ON LUN
IE.SNC	335	35.	FILE ID, FILE NUMBER CHECK
IE.SQC	334	36.	FILE ID, SEQUENCE NUMBER CHECK
IE.NLN	333	37.	NO FILE ACCESSED ON LUN
IE.CLO	332	38.	FILE WAS NOT PROPERLY CLOSED
IE.NBF	331	39.	OPEN - NO BUFFER SPACE AVAILABLE FOR FILE
IE.RBG	330	40.	ILLEGAL RECORD SIZE
IE.NBK	327	41.	FILE EXCEEDS SPACE ALLOCATED, NO BLOCKS
IE.ILL	326	42.	ILLEGAL OPERATION ON FILE DESCRIPTOR BLOCK
IE.BTP	325	43.	BAD RECORD TYPE
IE.RAC	324	44.	ILLEGAL RECORD ACCESS BITS SET
IE.RAT	323	45.	ILLEGAL RECORD ATTRIBUTES BITS SET
IE.RCN	322	46.	ILLEGAL RECORD NUMBER - TOO LARGE

```
  ...
IE.2DV  320     48.     RENAME - 2 DIFFERENT DEVICES
IE.FEX  317     49.     RENAME - NEW FILE NAME ALREADY IN
                        USE
IE.BDR  316     50.     BAD DIRECTORY FILE
IE.RNM  315     51.     CAN'T RENAME OLD FILE SYSTEM
IE.BDI  314     52.     BAD DIRECTORY SYNTAX
IE.FOP  313     53.     FILE ALREADY OPEN
IE.BNM  312     54.     BAD FILE NAME
IE.BDV  311     55.     BAD DEVICE NAME
IE.BBE  310     56.     BAD BLOCK ON DEVICE
IE.DUP  307     57.     ENTER - DUPLICATE ENTRY IN DIRECTORY
IE.STK  306     58.     NOT ENOUGH STACK SPACE
IE.FHE  305     59.     FATAL HARDWARE ERROR ON DEVICE
IE.NFI  304     60.     FILE ID WAS NOT SPECIFIED
IE.ISQ  303     61.     ILLEGAL SEQUENTIAL OPERATION
IE.EOT  302     62.     END OF TAPE DETECTED
IE.BVR  301     63.     BAD VERSION NUMBER
IE.BHD  300     64.     BAD FILE HEADER
IE.OFL  277     65.     DEVICE OFF LINE
IE.BCC  276     66.     BLOCK CHECK OR CRC ERROR
  ...
IE.NNN  274     68.     NO SUCH NODE
IE.NFW  273     69.     PATH LOST TO PARTNER
IE.BLB  272     70.     BAD LOGICAL BUFFER
IE.TMM  271     71.     TOO MANY OUTSTANDING MESSAGES
IE.NDR  270     72.     NO DYNAMIC SPACE AVAILABLE
IE.CNR  267     73.     CONNECTION REJECTED
IE.TMO  266     74.     TIMEOUT ON REQUEST
IE.EXP  265     75.     FILE EXPIRATION DATE NOT REACHED
IE.BTF  264     76.     BAD TAPE FORMAT
IE.NNC  263     77.     NOT ANSI 'D' FORMAT BYTE COUNT
IE.NNL  262     78.     NOT A NETWORK LUN
IE.NLK  261     79.     TASK NOT LINKED TO SPECIFIED ICS/ICR
                        INTERRUPTS
IE.NST  260     80.     SPECIFIED TASK NOT INSTALLED
IE.FLN  257     81.     DEVICE OFFLINE WHEN OFFLINE REQUEST
                        WAS ISSUED
IE.IES  256     82.     INVALID ESCAPE SEQUENCE
IE.PES  255     83.     PARTIAL ESCAPE SEQUENCE
IE.ALC  254     84.     ALLOCATION FAILURE
IE.ULK  253     85.     UNLOCK ERROR
  ...
IE.DRV  234     100.    DRIVER NOT FOUND IN ACP
IE.ACP  233     101.    DRIVER NOT INSTALLED

Errors from Group 1:

IE.UPN  377     129.    INSUFFICIENT DYNAMIC STORAGE
IE.INS  376     130.    SPECIFIED TASK NOT INSTALLED
IE.PTS  375     131.    PARTITION TOO SMALL FOR TASK
IE.UNS  374     132.    INSUFFICIENT DYNAMIC SPACE FOR SEND
IE.ULN  373     133.    UN-ASSIGNED LUN
IE.HWR  372     134.    DEVICE HANDLER NOT RESIDENT
IE.ACT  371     135.    TASK NOT ACTIVE
IE.ITS  370     136.    DIRECTIVE INCONSISTENT WITH TASK
                        STATE
```

IE.FIX	367	137.	TASK ALREADY FIXED/UNFIXED
IE.CKP	366	138.	ISSUING TASK NOT CHECKPOINTABLE
IE.TCH	365	139.	TASK IS CHECKPOINTABLE
...			
IE.RBS	361	143.	RECEIVE BUFFER IS TOO SMALL
...			
GE.IOR	277	193.	COMMAND I/O ERROR
GE.OPR	276	194.	INDIRECT FILE OPEN FAILURE
GE.BIF	275	195.	INDIRECT COMMAND SYNTAX ERROR
GE.MDE	274	196.	INDIRECT FILE DEPTH EXCEEDED
...			
IE.AST	260	208.	DIRECTIVE ISSUED/NOT ISSUED FROM AST
...			
IE.ALG	254	212.	ALIGNMENT ERROR
IE.WOV	253	213.	ADDRESS WINDOW ALLOCATION OVERFLOW
IE.NVR	252	214.	INVALID REGION ID
IE.NVW	251	215.	INVALID ADDRESS WINDOW ID
IE.ITP	250	216.	INVALID TI PARAMETER
IE.IBS	247	217.	INVALID SEND BUFFER SIZE (.GT.255.)
IE.LNL	246	218.	LUN LOCKED IN USE
IE.IUI	245	219.	INVALID UIC
IE.IDU	244	220.	ILLEGAL DEVICE OR UNIT
IE.ITI	243	221.	INVALID TIME PARAMETER
IE.PNS	242	222.	PARTITION/REGION NOT IN SYSTEM
IE.IPR	241	223.	INVALID PRIORITY (.GT. 250.)
IE.ILU	240	224.	INVALID LUN
IE.IEF	237	225.	INVALID EVENT FLAG (.GT. 64.)
IE.ADP	236	226.	PART OF DPB OUT OF USER'S SPACE
IE.SDP	235	227.	DIC OR DPB SIZE INVALID

NOTE

The symbol values given are truncated to byte length. The full word value is the sign extension of the byte value.

APPENDIX B

Examples

This appendix gives an example of how to create a simple DIOS user task. In order are given the assembler source code, the command file used to build the task, and the sequence of steps needed to run the task.

B.1 Example of a Macro Program using DIOS

All steps in the following program are explained in the comments.

```
        .TITLE  IOTASK -- ILLUSTRATE USE OF DIOS
;
; IOTASK -- SAMPLE PROGRAM TO ILLUSTRATE USE OF DIOS I/O
;
; DEFINE MACROS:
;
        .MCALL  WTSE$S,EXIT$S,ALUN$C    ;RSX DIRECTIVES
        .MCALL  QIO$S,QIOW$S
        .MCALL  LOAD,UNLOAD             ;DIOS OPERATIONS
        .MCALL  MCB,MCBE                ;MCB ALLOCATION
        .MCALL  MOOP$S,MOUT$S           ;MESSAGE OUTPUT
;
; DEFINE MACRO TO PRINT MESSAGE ON TERMINAL:
;
.MACRO  PRINT,LMSG,PMSG
        MOV     LMSG,PRM
        MOV     PMSG,PRM+2
        MOUT$S  #MO,#MFLG,,#FMT,#3,#PRM
        WTSE$S  #MFLG
.ENDM   PRINT
;
; DEFINE SYMBOLS:
;
        .MCALL  MCBDF$,DIODF$
        MCBDF$          ;DEFINE MCB OFFSETS
        DIODF$          ;DEFINE I/O FUNCTION AND ERROR CODES
        .PSECT          ;REENTER NORMAL P-SECTION
```

```
        $LDR == 3           ;DEFINE SYMBOL FOR LOADER LUN
        LUN = 4             ;MODULE LUN
        FLG = 4             ;EVENT FLAG
        MO = 2              ;MESSAGE OUTPUT LUN
        MFLG = 2            ;MESSAGE OUTPUT EVENT FLAG
;
; DEFINE MCB FOR MODULE TO BE LOADED:
;
MCB:
        MCB     1,1,0,0,0           ;FIRST PART
        MCBE    <FK>,1,,<FK>,MC.INT,,,,300,5,167770
;
; THIS MCB DEFINES THE FOLLOWING MODULE:
;
; MODULE TYPE:  "FK" -- FUNCTION KEYBOARD - DR11-C INTERFACE
; UNIT NUMBER:  1
; DRIVER ACP:   TASK NAME FK....
; CONTROL BITS: MC.INT = 1: REQUIRES INTERRUPT SERVICE
;               MC.CAM = 0: NOT A CAMAC MODULE
;               MC.SUB = 0: NOT A SUBMODULE
;
; INTERRUPT VECTOR ADDRESS:  300
; INTERRUPT PRIORITY:  5
; UNIBUS DEVICE ADDRESS: 767770 (16 BITS: 167770)
;
; EXECUTABLE CODE:
;
; PROGRAM DOES DIOS OPERATION INDICATED IN COMMENT.
; IF DIRECTIVE ERROR OCCURS, PRINTS
;
;       DIRECTIVE ERROR
;       <SOURCE>
;       <ERROR MESSAGE -- SEE APPENDIX A.>
;
; IF I/O ERROR OCCURS, PRINTS:
;
;       I/O ERROR
;       <SOURCE>
;       <ERROR MESSAGE -- SEE APPENDIX A.>
;
; <SOURCE>= "ALUN$S -","IO.ACW -", ETC. DEPENDING ON
; WHICH OPERATION CAUSED ERROR.
;
; IF NO ERROR, PRINTS:
;
;       WAITING FOR IO.RVB
;
; AND EXITS AS SOON AS 10 FUNCTION CODES ARE ENTERED ON
; FUNCTION KEYBOARD.
;
```

```
IOTASK:
        MOOP$S  #MO,#MFLG       ;ACCESS MESSAGE OUTPUT
;
; ASSIGN LUN TO LOADER DEVICE:
;
        ALUN$C  $LDR,DA,0       ;ASSIGN LOADER LUN
                                ; USE -$C FORM, WHICH IS
 FASTER
        BCC     1$              ;IF CC, DIRECTIVE OK
        JSR     R5,ALDIR        ;IF CS, REPORT ERROR
        .WORD   ALMSG           ;MESSAGE FOR "ASSIGN LUN"
1$:
;
; ACCESS FILE ON LOADER:
;
        QIOW$S  #IO.ACW,#$LDR,#FLG,,#STS ;ISSUE I/O REQUEST
        BCC     2$              ;IF CC, DIRECTIVE OK
        JSR     R5,ACDIR        ;IF CS, DIRECTIVE ERROR
        .WORD   ACMSG           ;MESSAGE FOR "ACCESS LOADER"
2$:     TSTB    STS             ;I/O ERROR OCCURRED?
        BPL     3$              ;IF PL, I/O OK
        JSR     R5,ACIOE        ;IF MI, I/O ERROR
        .WORD   ACMSG
3$:
;
; LOAD MODULE DEFINED IN MCB:
;
        LOAD    #LUN,#MCB,#STS  ;LOAD MODULE:
                                ;IF EVENT FLAG NOT SPECIFIED
                                ;USES FLAG 32. BY DEFAULT.
        BCC     4$              ;IF CC, DIRECTIVE OK
        JSR     R5,LDDIR        ;IF CS, DIRECTIVE ERROR
        .WORD   LDMSG
4$:     TSTB    STS             ;I/O ERROR OCCURRED?
        BPL     5$              ;IF PL, NO
        JSR     R5,LDIOE        ;IF MI, I/O ERROR FROM LOAD
        .WORD   LDMSG
5$:
;
; PERFORM I/O OPERATIONS ON MODULE:
;
; IO.INI -- INITIALIZE MODULE:
; PASS PARAMETERS IN INIBF.  LENGTH = INIBL BYTES
;
        QIOW$S  #IO.INI,#LUN,#FLG,,#STS,,<#INIBF,#INIBL>
        BCC     6$              ;IF CC, DIRECTIVE OK
        JSR     R5,INDIR        ;IF CS, DIRECTIVE ERROR
        .WORD   INMSG
6$:     TSTB    STS             ;I/O ERROR OCCURRED?
        BPL     7$              ;IF PL, NO
        JSR     R5,INIOE        ;IF MI, YES
        .WORD   INMSG
7$:
;
```

```
; IO.RVB -- READ DATA FROM MODULE:
; READ DATA INTO DATBF.  LENGTH = DATBL BYTES.
;
        QIO$S   #IO.RVB,#LUN,#FLG,,#STS,,<#DATBF,#DATBL>
        BCC     8$                  ;IF CC, DIRECTIVE OK
        JSR     R5,RVDIR            ;IF CS, DIRECTIVE ERROR
        .WORD   RVMSG
8$:
;
; I/O IS NOW IN PROGRESS.  TASK CONTINUES EXECUTING,
; MUST EXPLICITLY WAIT FOR I/O TO FINISH.
;
        PRINT   #WATL,#WATM         ;PRINT MESSAGE: WAITING
        WTSE$S  #FLG                ;WAIT FOR I/O TO FINISH
        BCC     9$                  ;IF CC, OK
        JSR     R5,WTDIR            ;IF CS, DIRECTIVE ERROR
        .WORD   WTMSG
9$:
        TSTB    STS                 ;I/O ERROR OCCURRED?
        BPL     10$                 ;IF PL, NO
        JSR     R5,RVIOE            ;IF MI, YES
        .WORD   RVMSG
10$:
;
; IO.TER -- TERMINATE MODULE:
;
        QIOW$S  #IO.TER,#LUN,#FLG,,#STS,,<#0,#4>
        BCC     11$                 ;IF CC, OK
        JSR     R5,TMDIR            ;IF CS, DIRECTIVE ERROR
        .WORD   TMMSG
11$:    TSTB    STS                 ;I/O ERROR OCCURRED?
        BPL     12$                 ;IF PL, NO
        JSR     R5,TMIOE            ;IF MI, YES
        .WORD   TMMSG
12$:
;
; UNLOAD MODULE:
;
        UNLOAD  #LUN,#STS,#FLG      ;SPECIFY EVENT FLAG
        BCC     13$                 ;IF CC, OK
        JSR     R5,ULDIR            ;IF CS, DIRECTIVE ERROR
        .WORD   ULMSG
13$:    TSTB    STS                 ;I/O ERROR?
        BPL     14$                 ;IF PL, NO
        JSR     R5,ULIOE            ;IF MI, YES
        .WORD   ULMSG
14$:
;
```

```
; DEACCESS LOADER:
;
        QIOW$S  #IO.DAC,#$LDR,#FLG,,#STS
        BCC     15$                 ;IF CC, OK
        JSR     R5,DCDIR            ;IF CS, DIR ERROR
        .WORD   DCMSG
15$:    TSTB    STS                 ;I/O ERROR?
        BPL     16$                 ;IF PL, NO
        JSR     R5,DCIOE            ;IF MI, YES
        .WORD   DCMSG
16$:    JMP     EXIT                ;ALL DONE, EXIT
;
; REPORT ERROR CONDITIONS WITH MESSAGE OUTPUT:
;
; REPORT DIRECTIVE ERRORS:
;
ALDIR:ACDIR:LDDIR:INDIR:RVDIR:TMDIR:ULDIR:DCDIR:WTDIR:
        PRINT   #DIRL,#DIRM         ;PRINT OUT "DIRECTIVE ERROR"
        MOVB    @#$DSW,R0           ;GET DIRECTIVE STATUS WORD
                                    ; (EXTEND SIGN)
        NEG     R0                  ;NEGATE TO GET RECORD NUMBER
        ADD     #128.,R0            ;ADD 128. BECAUSE DIRECTIVE
                                    ; ERRORS ARE FROM GROUP 1.
        BR      OUTMSG              ;OUTPUT MESSAGE
;
; REPORT I/O ERRORS:
;
ACIOE:LDIOE:INIOE:RVIOE:TMIOE:ULIOE:DCIOE:
        PRINT   #IOEL,#IOEM         ;PRINT OUT "I/O ERROR"
        MOVB    STS,R0              ;GET ERROR CODE FROM IOSB
                                    ; (EXTEND SIGN).
        NEG     R0                  ;NEGATE TO GET RECORD NUMBER
        TSTB    STS+1               ;IS IT GROUP 1?
        BEQ     1$                  ;IF EQ, NO
        ADD     #128.,R0            ;IF NE, YES: ADD 128. TO
                                    ;TO RECORD NUMBER.
1$:
;
OUTMSG:
        PRINT   #10.,(R5)+          ;PRINT SOURCE OF ERROR
        MOUT$S  #MO,#MFLG,,,,,R0    ;OUTPUT MESSAGE.
        WTSE$S  #MFLG               ;WAIT FOR COMPLETION
EXIT:   EXIT$S                      ;EXIT TASK
;
; DATA:
;
; DATA BUFFER FOR READ VIRTUAL BLOCK:
;
DATBF:  .BLKW   5                   ;READ 5 WORDS
DATBL=.-DATBF                       ;SET LENGTH = 10. BYTES.
;
; I/O STATUS BLOCK FOR MODULE:
;
STS:    .BLKW   2                   ;RESERVE 2 WORDS
;
```

```
; BUFFER FOR IO.INI:
;
; FUNCTION KEYBOARD (FK) REQUIRES NO DATA ON INIT:
; HOWEVER, A DUMMY BUFFER IS RESERVED HERE TO ILLUSTRATE
; NORMAL USE.
;
INIBF:  .BLKW   2                       ;RESERVE 2 WORDS
INIBL=.-INIBF                           ;LENGTH OF INIT BUFFER
;
; MESSAGE BUFFERS FOR PRINT:
;
.MACRO  LINE,TEXT
        .BYTE   12
        .ASCII  >TEXT>
        .BYTE   15
        .EVEN
.ENDM   LINE
DIRM:   LINE    <DIRECTIVE ERROR>
DIRL=.-DIRM
.EVEN
;
IOEM:   LINE    <I/O ERROR>
IOEL=.-IOEM
.EVEN
;
WATM:   LINE    <WAITING FOR IO.RVB>
WATL=.-WATM
.EVEN
;
; I/O FUNCTION CODE IDENTIFYING MESSAGES:
;
ALMSG:  LINE    <ALUN$C ->
ACMSG:  LINE    <IO.ACW ->
LDMSG:  LINE    <IO.LOD ->
INMSG:  LINE    <IO.INI ->
RVMSG:  LINE    <IO.RVB ->
TMMSG:  LINE    <IO.TER ->
ULMSG:  LINE    <IO.UNL ->
DCMSG:  LINE    <IO.DAC ->
WTMSG:  LINE    <WTSE$S ->
.EVEN
;
; FORMAT BUFFER FOR PRINT:
;
FMT:    .ASCIZ  /%VA/
.EVEN
;
PRM:    .BLKW   0                       ;PARAMETER LIST
.END    IOTASK
```

B.2 Task Builder Command File

To build the task, type

```
>TKB @IOTASK.CMD
```

The contents of IOTASK.CMD are:

```
; COMMAND FILE TO BUILD I/O TASK EXAMPLE:
;
IOTASK,IOTASK=IOTASK
/
ASG=MO:1          ;ASSIGN MESSAGE OUTPUT
UNITS=3           ;3 LUNS ARE REQUIRED
;
; NOTE:  DA0 IS ASSIGNED BY ALUN$ DIRECTIVE, NOT
;        WHEN TASK IS BUILT.
;
//
```

B.3 Running the I/O Task

* Preparing the DIOS and Message Output Systems:

These commands are only done once after starting RSX. If RSX is saved in this state, these commands may be omitted.

```
>@[1,2]INSDIOS
>INS [1,54]LOA
>LOA MO:
>REM ...LOA
>MOU MO0:
>INS [1,54]DADRV
>SET /NOMAIN=DAPAR
>SET /MAIN=DAPAR:740:15:TASK
>RUN [1,54]DIOLNK
>INS [1,54]LDRACP
>INS [1,54]FKACP
>MOU DA0:
```

* Running the task with DIOS installed:

```
>MOU MO0:                (Only if MO: not mounted)
>MOU DA0:                (Only if DA0: not mounted)
>RUN IOTASK
>DMO DA0:
>DMO MO0:
```

APPENDIX C

Macro Expansions

C.1 LOAD

```
.MACRO  LOAD,LUN,MCB,STS,FLG
        .MCALL  QIOW$S
.IF B,<FLG>
        QIOW$S    IO.LOD, $LDR, 32.,,STS,,<MCB, M.DDP,LUN>
.IFF
        QIOW$S    IO.LOD, $LDR,FLG,,STS,,<MCB, M.DDP,LUN>
.ENDC
.ENDM   LOAD
```

C.2 UNLOAD

```
.MACRO  UNLOAD,LUN,STS,FLG
.IF B,<FLG>
        QIOW$S    IO.UNL,LUN, 32.,,STS
.IFF
        QIOW$S    IO.UNL,LUN,FLG,,STS
.ENDC
.ENDM
```

C.3 DIODF$ -- Function and Error Codes

```
.MACRO  DIODF$,B
; FUNCTION CODES
;
Y       = 400           ;BYTE FACTOR
IO.MNT  ='B  5.*Y       ;MOUNT DA0
IO.DMO  ='B  6.*Y       ;DISMOUNT DA0
IO.CKP  ='B 17.*Y+1     ;READ WITH CHECKPOINTING REQUEST
IO.LOD  ='B 18.*Y       ;LOAD MODULE
IO.INI  ='B IO.LOD+1    ;INITIATE FUNCTION
IO.TER  ='B IO.LOD+2    ;TERMINATE FUNCTION
IO.RRD  ='B IO.LOD+3    ;READ IN RANDOM MODE (MUX)
IO.XAK  ='B IO.RRD+1    ;WRITE ACK (REMOTE)
IO.XNK  ='B IO.RRD+2    ;WRITE NAK (REMOTE)
IO.XCN  ='B IO.RRD+3    ;WRITE CAN (REMOTE)
```

```
IO.WTP  ='B IO.RRD+4     ;WRITE TRANSPARENT (REMOTE)
IO.RTP  ='B IO.RRD+5     ;READ TRANSPARENT (REMOTE)
IO.LD2  ='B 32.*Y        ;FINISH LOADING MODULE (DRIVER ACP)
IO.ULM  ='B 16.*Y        ;UNLOAD MODULE (DRIVER ACP)
IO.UL2  ='B 33.*Y        ;FINISH UNLOADING MODULE(LOADER ACP)
IO.ABL  ='B IO.UL2+1     ;ABORTED LOAD DETECTED BY DRIVER ACP
                         ;AND PROCESSED BY LOADER.
;
;NONSTANDARD ERROR CODES:
;
        IE.DRV='B-100.   ;DRIVER NOT PRESENT IN ACP
        IE.ACP='B-101.   ;DRIVER ACP NOT INSTALLED.
        IE.IDS='B-137.   ;INCONSISTENT WITH DEVICE STATE
        IE.PWF='B-139.   ;OPERATION ABORTED DUE TO POWERFAIL
;
.ENDM   DIODF$
```

C.4 MCBDF$ -- MCB Offsets and Bits

```
; DEFINE MODULE CONTROL BLOCK OFFSETS:
;
.MACRO  MCBDF$,B,L
.ASECT
BT.MCB ='B 3     ;TYPE CODE FOR MCB
M.DID  ='B 3     ;OFFSET TO DIAGNOSTIC ID CODE
.ASECT
.=12
M.TYP:'L .BLKW  1        ;MODULE TYPE CODE
M.UNIT:'L .BLKB 1        ;UNIT NUMBER
M.MID:'L .BLKB  1        ;MODULE ID CODE
M.ACP:'L .BLKW  1        ;ACP - IDENTIFIER
M.CTL:'L .BLKW  1        ;CONTROL BITS
M.DLN:'L .BLKB  1        ;DATA WORD LENGTH
M.DFM:'L .BLKB  1        ;DATA FORMAT CODE
M.DCT:'L .BLKW  1        ;NUMBER OF DATA WORDS IN SHOT
M.DAT:'L .BLKW  1        ;RECORD   OF DATA BLOCK
M.VCT:'L .BLKB  1        ;INTERRUPT VECTOR ADDRESS/4
M.PRI:'L .BLKB  1        ;INTERRUPT PRIORITY IN BITS <5:'L7>
M.ADR:'L .BLKW  1        ;MODULE ADDRESS (CAMAC OR UNIBUS)
M.ERR:'L .BLKB  1        ;I/O ERROR CODE
M.LUN:'L .BLKB  1        ;MODULE LUN RETURNED FROM LOAD
         .BLKB  10       ;RESERVED LOCATIONS
M.LPM:'L .BLKW  1        ;NUMBER OF BYTES OF USER PARAMETERS
M.DDP:'L                 ;END OF MINIMAL MCB
;
; DEFINE MCB STATUSCONTROL BITS
;
MC.CTL='B 1      ;MODULE CONTROLS OTHER MODULES
MC.GEN='B 2      ;MODULE GENERATES DATA
MC.PRE='B 10     ;MODULE DELIVERS DATA BEFORE SHOT
MC.PST='B 20     ;MODULE DELIVERS DATA AFTER SHOT
MC.HLD='B 40     ;MODULE NOT TO BE UNLOADED TIL AFTER SHOT
MC.INT='B 100    ;MODULE REQUIRES INTERRUPT SERVICE
MC.CAM='B 200    ;MODULE IN CAMAC SYSTEM
MC.NPR='B 4000   ;MODULE CAPABLE OF DMA I/O
```

```
MC.OVF='B 40000 ;MODULE GENERATED MORE DATA THAN FIT IN FILE
MC.ERR='B 100000 ;ERROR WAS DETECTED ON MODULE DURING SHOT
;
        .MACRO  MCBDF$          ;REDEFINE MACRO FOR 2ND PASS
        .ENDM   MCBDF$
.ENDM   MCBDF$
```

NOTE

Expansions of MCB and MCBE macros are given in the GALE System Programmer's Handbook.

APPENDIX D

The CAMAC System

D.1 Logical Organization

The CAMAC system consists of one or more crates each outfitted with a crate controller interfaced to the PDP-11. The maximum number of crates is determined by the type of crate controlling system being used, as discussed below. Each crate has a logical crate number; the crates are numbered consecutively from 1 to the actual (or planned) number of crates in the system. The crate configuration is defined when DIOS is generated and may not be changed without regenerating DIOS.

Each crate has 25 slots, or stations, for placing plug-in modules. The uppermost m slots are reserved for the crate controller: usually m is 2, leaving stations 1 to 23 free. The free stations are available for DIOS modules.

Each module occupies one or more consecutive stations of a crate.

Simple modules consist of a single physical plug-in unit.

Composite modules may consist of separate plug-in units with complementary functions treated as a single module under DIOS. This implies that a single DIOS driver controls all of them. It is required that the components of a composite module occupy consecutive stations. The sub-structure (number and order of components) is either fixed for the module type, or must be passed to the driver by a special I/O function.

A LAM may be generated by a module when it requires immediate service. The CAMAC controller translates the LAM to an interrupt request on the Unibus. The module can theoretically generate a LAM on any of the stations it occupies: however, a strict requirement of DIOS is that the module generate a LAM on only one of these stations.

The position of a module within the CAMAC system is defined by giving the logical crate number of its crate, and a

station address within the crate. The station address must be one of the stations occupied by the module. Further rules governing the choice of station address are

-- If the module generates a LAM, the address must be the station number of the LAM.

-- If all CAMAC functions are directed to one of the stations within the module, that station is used as the address.

In general, the description of the module driver will specify which station relative to the leftmost one is to be used as the address.

The configuration of modules in the DIOS system is dynamic: modules may be removed and re-inserted into other crates. User tasks must of course change the module address in the MCB if this is done.

D.1.1 Specifying the Module Address

When the module is loaded, the module address is coded into M.ADR of the MCB as follows:

M.ADR Station address within crate (1 to 23)

M.ADR+1 Logical crate number

D.1.2 Specifying the Crate Configuration

The crate configuration is specified when DIOS is generated; the details are given in Appendix E. The following information specifies the configuration.

1. Number of Crates:
   This is the total number of crates presently connected or planned. For each logical crate from 1 to the number of crates specified, entries are allocated in the various tables defining the crate configuration.

   For each logical crate the following data are specified.

2. Controller Type:
   This identifies the type of crate controller system by which the crate is interfaced to the PDP-11. The present version of DIOS supports either the Borer or ICP-11 controller types; currently, all crates must have the same type.

3. Controller Width:
   This reserves the top m stations for the crate controller, allowing the remaining stations 1 to 25-m to be used for modules. In the current version of DIOS, only controllers of width 2 are supported.

4. Physical Crate Address:
   This information is used to determine which Unibus addresses are recognized by the crate controller system. The form of the addressing information depends on the controller system in use.

5. Interrupt Vector Area:
   This gives the base address of a block of interrupt vectors used by the controller system to dispatch LAM's from this crate to the proper interrupt service routine.

6. Interrupt Priority:
   This gives the hardware priority level of the LAM interrupts from this crate.

D.2 General Hardware Procedures

D.2.1 Crate Initialization

When DIOS is activated with MOU DA0:, the following sequence of actions occurs.

1. Reset Crate (Z) is executed.
2. Clear Crate (C) is executed.
3. Crate Inhibit (I) is cleared.
4. Interrupts from the crate are enabled.

It may be the case that a given crate has not been installed, or is switched offline. In this case, the following message is printed on the Console Log:

```
*** DA: -- CRATE n OFFLINE
```

The first time an attempt is made to load a module in this crate, DIOS again attempts to initialize the crate with the above procedure. If the crate has been switched online in the meantime, the initialization is successful and the module is loaded. Otherwise, the module is not loaded, and the error "DEVICE OFFLINE" is returned.

D.2.2 Crate Termination

When DIOS is deactivated with DMO DA:, the following sequence of actions occurs on each on-line crate.

1. Interrupts from the crate are disabled.
2. Crate Inhibit (I) is set.
3. Reset Crate (Z) is executed.
4. Clear Crate (C) is executed.

D.2.3 Interrupt Handling

When a LAM is received from a station in the crate, the Crate controller causes an interrupt to occur in the PDP-11 at the priority level of the crate. At this point, further interrupts from the crate are disabled. After the module driver finishes processing the interrupt, interrupts from the crate are reenabled by the CAMAC interrupt handler to allow further LAM's to be recognized. It is the responsibility of the module driver to clear the module LAM before exiting back to the CAMAC interrupt handler.

D.3 Specifics of the Borer System

In the Borer System, each crate is outfitted with a crate controller connected directly to the Unibus. Each controller has a unique set of addresses by which it is controlled; thus the crates operate independently of one another.

D.3.1 Borer Crate Configuration

1. Number of Crates:
   From 1 to 10 crates may be specified.
2. Controller Type:
   Borer PDP-11/CAMAC Crate Controller, Type 1533A
3. Controller Width:
   2

4. Physical Crate Address:

The Borer controller is addressed through a range of 1420(8) byte locations reserved in the device page. The base of this address range is jumpered into the controller interface; it is given as the controller address.

Standard Assignment of Base Addresses:

Crate 1	764000
Crate 2	766000
Crate 3	740000
Crate 4	742000
Crate 5	744000
Crate 6	746000
Crate 7	750000
Crate 8	752000
Crate 9	754000
Crate 10	756000

5. Interrupt Vector Area:
The base address of the interrupt vector block must be given as jumpered into the interrupt generator of the crate controller.

Standard (Pre-wired): Base = 200

Vector Block Description (Base = 200):

200	Q (X) Interrupt
204	LAM's 15 - 23
210	LAM 14
214	LAM 13
...	
274	LAM 1

6. Interrupt Priority:
The interrupt priority is wired into the interrupt generator of the crate. It may be 4, 5, 6, or 7.

D.3.2 Notes

1. LAM's 15 to 23 all cause an interrupt to the same vector location. The CAMAC interrupt dispatcher passes them to the proper module driver by reading a LAM bit pattern from the crate. Thus the module may be installed into any station and still have its interrupts processed. However, interrupts are serviced more quickly if the module LAM is from stations 1 - 14, or only one module with LAM is in stations 15 - 23 (in which case the LAM pattern is not read.

D.4 Specifics of the ICP System

The ICP system interfaces several crate controllers to the PDP-11 through one interface called a Branch Controller. The crate controllers are connected to the branch controller by a command and data bus called the Branch Highway. The branch highway, branch functions, and crate controller operate as specified in the ESONE report EUR 4100e.

Up to 7 crates may be connected to one branch. Each crate has a physical crate number from 1 to 7.

Up to 2 branch controllers may be connected to the Unibus.

D.4.1 ICP Crate Configuration

1. Number of Crates:
   From 1 to 7 crates with 1 branch, or from 2 to 14 with 2 branches may be specified.

2. Type of Controller:
   Schlumberger ICP-11A Crate Controller (Type A) or other Type A Crate Controller, connected to a Schlumberger ICP-11 or ICP-11A Branch Controller.

3. Width of Crate Controller:
   2

4. Physical Crate Address:
   The crate address is given by four parameters:

   * The number of the branch controller to which the crate is connected (1 or 2).

   * The physical crate number within the branch (1 - 7).

   * The base of the branch controller's CAMAC register area. This is a range of 1400(8) byte addresses on the device page used to address individual stations and subaddresses on all crates of the branch. The base address is wired into the branch controller, and is the same for all crates in the branch. Standard settings are:

Branch 1	764000
Branch 2	766000

   * Unibus address of branch controller CSR. This is the first in a 4-word group of device registers controlling the branch controller. The address is wired into the controller; it is the same for all crates in the branch.

Range: 766000 - 767776
Standard Setting: 766000

5. Interrupt Vector Area:
The base address of the vector area is wired into the branch controller. It is the same for all crates in the branch.

Standard Assignment: 400
Vector Allocation (Base = 400):

400	Graded LAM 24
404	Graded LAM 23
...	
534	Graded LAM 1
540	External Demand (Not Used)

6. Interrupt Priority:
The interrupt priority is wired in the branch controller and is the same for all crates in the branch.

Standard Setting: 4

D.4.2 Notes:

1. Individual module LAM's may be logically combined and the resulting signals output on Graded LAM lines 1 to 24. These Graded LAM's are associated with the vectors as in the table above. Under DIOS, the mapping must be 1 to 1 from the module LAM to Graded LAM: LAM 1 gives Graded LAM 1, etc.

2. On initialization and on interrupt the operation of enabling the crate interrupts involves two steps:

-- Enabling the Crate Demand, by issuing a branch operation to the crate controller.

-- Enabling the Branch Demand, by setting a bit in the branch controller CSR.

Conversely, disabling interrupts from the crate on termination requires

-- Clearing the Branch Demand Enable bit in the Branch controller CSR.

-- Disabling the Crate Demand in the Crate Controller with a branch operation.

3. For each branch, only one set of interrupt vectors for LAM 1 to 24 is available. Thus modules loaded into the same station numbers on different crates of the same branch will interrupt to the same vector location. The CAMAC interrupt dispatcher determines from which crate of the branch the interrupt came, and calls the appropriate driver interrupt routine.

4. On specifying the Branch Controller Unibus addresses and vector area, once any crate from a branch has been defined, the rest of the crates from that branch must have the same values for these parameters. This is automatically done in the DIOS generation process.

APPENDIX E

Generating a DIOS System

E.1 Introduction

In order to apply the DIOS I/O procedures discussed in this manual, a working DIOS system must be generated on the RSX system disk. This involves

* Specifiying the characteristics of the DIOS system to be generated.

* Building the component tasks on the system disk.

* Transferring other associated data files to the system disk.

The files needed to generate DIOS are distributed on one of the following media:

RK05 Disk Cartridge

9-Track Magnetic Tape

DECtape

Among the distributed files are indirect command files which direct the DIOS generation process.

To generate DIOS, the user must first copy the distributed files to a Files-11 device (preferably a disk), known as the distribution device. The distribution device and the system disk on which DIOS is to run (target system) are then mounted. The generation is accomplished by executing a command file on the distribution disk: first the characteristics of the DIOS system are determined by answering a series of questions asked by the command file; then the component tasks and other files are created; finally the target system is started and the completed DIOS system is installed.

E.1.1 Working Components of DIOS

This section lists the various files on the system disk containing the DIOS components; these files are created or modified in the course of generating DIOS.

[1,54]DADRV.TSK contains the system-resident portion of DIOS. Its contents are installed into partition DAPAR within the region of memory reserved for the executive. It has the following subcomponents:

* Data tables for loader device DA0:
* Device driver for DA:
* DIOS interrupt dispatcher
* Data tables for CAMAC crate configuration
* CAMAC interrupt dispatcher
* CAMAC function processor

[1,54]DADRV.STB is the symbol table defining the entry points of routines contained in DADRV. It is required for the generation of new driver ACP's.

[1,54]DIODEF.CMD defines the DIOS system characteristics for the command file used to generate driver ACP's.

[1,54]LDRACP.TSK is the task image of the loader ACP. It must be installed in order to activate and use DIOS.

[1,54]xxACP.TSK is the task image of a representative driver ACP, installed as task xx.... . Any number of driver ACP's may be generated. xx stands for the 2-character ACP identifier.

[1,54]DIOLNK.TSK is a utility task which is run to link the loader device DA0: into the RSX device list when DIOS is installed.

[1,2]INSDIOS.CMD is a command file which performs the steps needed to install and activate DIOS when it is executed.

[1,54]MODRV.TSK is the task image of the loadable driver for MO:, the Message Output device.

[1,54]MODRV.STB is the symbol table required to load MO:.

[1,54]MOACP.TSK is the Message Output ACP associated with device MO0:.

[1,2]MOTFMT.MSG is the standard error message file containing the error messages discussed in Appendix A.

[1,2]USRFMT.MSG is a user format file required to mount the Message Output processor. It is initialized to contain user-defined messages by the Message Output generation procedure.

[1,54]DIOMOU.TSK is a modified version of the MCR MOU command processor; this task is required to mount the devices DA0: and MO0:.

[1,1]RSXMAC.SML, the System Macro Library, contains the macros discussed in Chapter 2 and Appendix B. It also contains the Message Output macros described in the GALE Programmer's Handbook.

[1,1]SYSLIB.OLB, the System Object Library, contains the global symbol definitions of DIOS I/O functions, error codes, and MCB offsets as defined in the modules MCBSYM and DIOSYM.

E.1.2 Prerequisites

E.1.2.1 Target System Prerequisites -

The installation under which DIOS 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 CAMAC systems supported by DIOS (see Appendix D).

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

   ```
   * 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 DEVICE INTERRUPT VECTOR? [0]:
   ```

   The address given must be greater than or equal to the highest interrupt vector to be used by a DIOS module or CAMAC crate. For CAMAC crates, the top of the interrupt vector region must lie within this range.

   ```
   * 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 size of the system pool must be at least 2K bytes.

NOTE

If the host RSX system does not meet these specifications, a new system must be generated.

E.1.2.2 Generation System Prerequisites -

If DIOS is to be generated on the target installation, a second Files-11 device must be available to contain the distributed files.

If DIOS is to be generated on another installation, additional drives must be available to mount the target system and distribution devices.

If a seperate drive for the distribution device is not available, the generation may still be performed if there is enough room on either the installation or target system disk to hold the distributed files plus any scratch files generated. In this case, the distribution device is identical to one of the system devices.

The target system device must have 140 free blocks to contain the task images and other files created during the system generation. The distribution device must have 40 free blocks to contain temporary command and MACRO-11 files created.

E.1.3 Distribution Device Contents

This section lists the files required to perform a DIOS generation. They must be copied from the original distribution medium onto the distribution device, which is a Files-11 volume (preferably a disk). The directories and file names listed are as they appear on the copied distribution device.

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

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
DIOMOU.ODL	Overlay description for DIOS MOU processor

UIC [1,210] contains object files, libraries, 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
LDRACP.OLB	Loader ACP components
DRVACP.OLB	Driver ACP components Distributed DIOS module drivers
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
MOTFMT.MSG	System Message File for Message Output
USRFMT.MSG	Dummy User Message File for Message Output

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

CAMTBL.ICP	Define ICP crate tables
CAMTBL.BOR	Define Borer crate tables
DIOSMAC.MAC	DIOS macro definitions for RSXMAC.SML
MOUTMAC.MAC	Message Output macros for RSXMAC.SML
PDEMAC.MAC	General PDE macros for RSXMAC.SML

E.1.4 Preparing for DIOS Generation

Before beginning to generate a DIOS system, the following preparations must be made.

* Collect the following information describing the target system and the intended DIOS system:

1. Size of the executive region, either 16K or 20K. The size is 20K if the question
   DO YOU WANT SUPPORT FOR A 20K EXECUTIVE? [Y/N]:
   was answered yes when the RSX system was generated.

2. Address of the highest device interrupt vector, as specified when the RSX system was generated.

3. Number of the 100-byte block on which partition DAPAR begins.

4. Number of terminals (devices of type TTn:) in the target system, including those marked "OFFLINE".

5. Unibus addresses, vector area addresses, and interrupt priorities of all CAMAC crate or branch controllers, as called for in Appendix D.

6. Types of modules desired, object file specifications of any user-written drivers, and organization of drivers into driver ACP's.

* If you are generating DIOS at the installation it is to run on, boot the target RSX system disk. If you are generating DIOS under another RSX11M V 3.0 system, mount the target system on a second disk drive. The target system must be write enabled.

* Mount the distribution device containing the copied distribution files on an available drive, with write enabled.

* In a multi-user system, log on under a privileged account. Set the terminal UIC to [1,200].

* Start the DIOS generation process by typing

```
@ddn:[1,200]DIOSGEN
```

where ddn is the distribution device.

The command file will guide you through the generation procedure by printing comments and asking question on the user terminal; these are listed and explained in the next section.

E.2 Details of Generation Procedure

This section describes the execution of the command file [1,200]DIOSGEN.CMD and subsidiary command files. The questions and printed comments are listed in upper case, followed by explanatory text in mixed case if needed.

The following should be noted:

* MCR commands of the form INS [1,54]xxx and REM [1,54]xxx are often omitted if the corresponding task xxx was already installed. They are simply listed in the following without comment.

* <CR> stands for "Carriage Return", entered by typing the RETURN key on the terminal.

E.2.1 Assigning Devices

```
; GENERATE THE DIOS SYSTEM:
;
; THE FOLLOWING SERIES OF QUESTIONS DEFINE THE CHARACTERIS-
; TICS OF THE DIOS SYSTEM BEING GENERATED.
; 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]:
  When you have read the text, continue.

; ASSIGN DEVICES:
;
; "DISTRIBUTION DEVICE" IS THE DEVICE CONTAINING THIS COM-
; MAND FILE AND ALL OTHER COMMAND, OBJECT, AND MACRO FILES
; NEEDED TO GENERATE DIOS.
;
; "TARGET SYSTEM DEVICE" IS THE DEVICE CONTAINING THE RSX
; SYSTEM UNDER WHICH THE DIOS SYSTEM BEING GENERATED IS TO
; RUN.
;
```

```
; "MAP DEVICE" IS THE DEVICE ON WHICH ALL TASK-BUILDER MAPS
; WILL BE GENERATED.

* INPUT THE DISTRIBUTION DEVICE [DDU:] <DK1:> [S]:
```

Type the device name and unit number of the Files-11 volume containing the distributed files. DK1: is the default.

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

Type the device name and unit number of the disk containing the target system. Default is SY0:, the currently running system disk.

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

If no map device is given, no maps will be produced. If one is given, you will be asked whether a map is wanted for each individual component to be built. It is recommended to specify a disk if possible, as this speeds up task building.

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

This question is not asked if no map device was specifed, or if the print spooler task (PRT...) is not installed. Answer no if the map device is a disk and you wish to suppress the automatic spooling of map files.

```
ASN <dis>=DD:
ASN <sys>=TS:
ASN <map>=MP:
```

The distribution device, target system device, and map device are assigned to DD:, TS:, and MP: respectively. The map device is not assigned if unspecified.

```
PIP @DD:[1,200]CLEANUP
```

The scratch files created by the last DIOS generation using this distribution device are purged and renamed to version 1.

E.2.2 Selecting Options

```
* HAS A DIOS SYSTEM BEEN GENERATED BEFORE ON ddn:? [Y/N]:
```

Answer no if you are generating DIOS for the first time on this target system. Answer yes if you are implementing updates to a previously generated DIOS system. "ddn" denotes the target system device.

```
* DO YOU NEED TO GENERATE THE MESSAGE OUTPUT PROCESSOR? [Y/N]:
```

The Message Output processor is required by DIOS and user tasks: if it has not previously been generated, answer yes. Also answer yes if you are implementing updates to MO. If you answer no, MO will not be generated.

```
* DO YOU WANT TO UPDATE THE DIOS MOUNT PROCESSOR? [Y/N]:
```

The DIOS Mount processor is required by both the basic DIOS system and the Message Output processor: if neither

was previously generated, this question is not asked and DIOMOU is generated. Otherwise, answer yes to implement updates to DIOMOU. If you answer no, DIOMOU is not generated.

If DIOS is being generated for the first time, all components must be generated, and the following questions are skipped. Otherwise, select which components need to be updated by answering yes to the corresponding question:

* DO YOU WANT TO UPDATE SYSTEM AND MACRO LIBRARIES? [Y/N]:
  Answer yes only if changes to the DIOS macros, symbol definitions or library subroutines are being implemented.

* DO YOU WANT TO UPDATE THE SYSTEM-RESIDENT ROUTINES? [Y/N]:
  If you answer yes, the system-resident components in DADRV.TSK will be generated, including the structures and driver associated with DA0: and the CAMAC configuration.

* DO YOU WANT TO UPDATE THE LOADER ACP? [Y/N]:
  If you answer yes, the loader ACP will be rebuilt.

* DO YOU WANT TO GENERATE ANY DRIVER ACP'S [Y/N]:
  If you answer yes, the driver ACP's to be installed with DIOS will be specified and built.

** WARNING **

If you modify the system-resident routines, you must also rebuild the loader and driver ACP's. Attempting to use the old versions will cause a system crash.

E.2.3 Specifying the System-Resident Routines

This section is skipped if you are not generating the system-resident routines.

Otherwise, DIOSGEN executes the command file DD:[1,200]DADRVGEN.CMD to issue the questions and comments in this section. The results are used to create the following files:

DD:[1,200]DADRV.BLD is the task-build command file for building DADRV.TSK, DADRV.STB, and DIOLNK.TSK.

TS:[1,54]DIODEF.CMD is a command file used by other command files to remember general system parameters.

```
; GENERATE THE DIOS SYSTEM-RESIDENT ROUTINES:
;
* DO YOU WANT A MAP? [Y/N]
```

This question is asked only if a map device was specified. If you answer yes, the map files DADRV.MAP and DIOLNK.MAP will be created.

```
; DEFINE SYSTEM CHARACTERISTICS:
* INPUT SIZE OF RSX SYSTEM [16K OR 20K] <16K> [S]:
```

Type 16K or <CR> if the target RSX system has a resident executive smaller than 16K words. Type 20K only if the question "DO YOU WANT SUPPORT FOR A 20K EXECUTIVE?" was answered yes when the target RSX system was generated.

```
* INPUT THE HIGHEST INTERRUPT VECTOR ADDRESS <774> [S]:
```

Input the octal address of the highest device interrupt vector which may be specified in a module being loaded. This should be equal to the address specified for the highest device interrupt vector in the RSX system generation. The default is 774 .

```
* INPUT STARTING BLOCK OF PARTITION DAPAR [? IF UNKNOWN] [S]:
```

Input the number of the 100-byte block on which partition DAPAR begins; this is equal to the byte address divided by 100(8). It must be an octal integer between 0 and 763 (16K executive) or 1163 (20K executive). If this number is unknown, type a "?" to request a display of the target system's partitions, as follows:

```
; HERE ARE THE PARTITIONS PRESENTLY IN THE TARGET SYSTEM:
; NOTE THE BASE ADDRESS OF DAPAR AND DIVIDE IT BY 100(8)
; TO CALCULATE THE STARTING BLOCK.
INS [1,54]VMR
VMR @DD:[1,200]PAR
LDR     000000 000000 MAIN TASK
DAPAR   074000 001500 MAIN COM
   ...
(Further partitions in the system are displayed.)
   ...
REM ...VMR
* DOES PARTITION "DAPAR" EXIST? [Y/N]
```

If DAPAR appeared in the above list, answer yes and note the base address in the first column of numbers. The question "INPUT STARTING BLOCK ..." will be repeated.
If DAPAR did not appear, you may answer yes <u>if</u> you know the address at which you intend to install the partition.

Otherwise, answer no: the DIOS generation will exit by printing a suggestion as to how to make room for and include the DAPAR partition. When the partition has been installed, you may start the DIOS generation over.

E.2.4 Specifying the CAMAC Configuration

* DO YOU HAVE A CAMAC SYSTEM? [Y/N]:
If you answer no, the rest of this section defining the CAMAC system will be skipped, and a DIOS system will be created which does not support any CAMAC modules.

If you answer yes, the command file DADRVGEN.CMD executes the command file DD:[1,200]CAMACGEN.CMD, which asks the questions in the rest of this section. The results are used to create one of two MACRO-11 files:

DD:[1,220]CAMDEF.ICP is created if an ICP-11 system was specified.

DD:[1,220]CAMDEF.BOR is created if a Borer system was specified.

The appropriate file is assembled along with CAMTBL.ICP or CAMTBL.BOR to allocate and initialize data tables describing the CAMAC configuration. These tables are then linked into DADRV.TSK.

The CAMAC configuration is set up according to the principles discussed in Appendix D, which must be consulted along with this section to determine how the questions should be answered.

The logical crates are defined in numerical order starting with crate number 1. At least one crate must be defined. After defining each crate, the operator is asked if there are more crates in the configuration; if so, the questions defining a crate are repeated, until all crates have been specified.

A simplified procedure is provided for defining an ICP-11 system. The present version of DIOS does not support the general ICP configuration; furthermore, only one branch is supported. Thus the short form is the only one presently allowed.

```
; DEFINE THE CAMAC CRATE CONFIGURATION:
;
; DEFINE CAMAC CRATE # 1:
* WHAT IS THE CONTROLLER TYPE [ICP OR BORER] [S]:
```

Answer "ICP" if you have a Schlumberger ICP-11 multi-crate branch controller. Answer "BORER" if the crate has a Borer single-crate controller. Once you specify the type for the first crate, the remaining crates must have the same type of controller; hence this question is omitted for the remaining crates. Subsequent questions depend on the controller type.

E.2.4.1 Defining an ICP-11 Configuration -

The following questions are asked if you specified an ICP controller.

```
; THE GENERAL METHOD OF GENERATING THE ICP CAMAC SYSTEM AL-
; LOWS ANY VALID LOGICAL CRATE NUMBER TO BE ASSIGNED TO A
; CRATE WITH A GIVEN BRANCH NUMBER AND PHYSICAL CRATE AD-
; DRESS.
; A SHORTER METHOD EXISTS WHICH REQUIRES ONLY THE NUMBER OF
; CRATES ON BRANCH 1 AND BRANCH 2 TO BE SPECIFIED.  THE LOG-
; ICAL CRATE NUMBERS ARE THEN ASSIGNED AS FOLLOWS:
;
;  LOGICAL CRATES 1:M = PHYSICAL CRATES 1:M ON BRANCH 1.
;  LOGICAL CRATES M+1:M+N = PHYSICAL CRATES 1:N ON BRANCH 2.
;
; WHERE M AND N ARE THE NUMBER OF CRATES ON BRANCH 1 AND 2
; RESPECTIVELY.  AT LEAST ONE CRATE MUST BE SPECIFIED ON
; BRANCH 1.
;
; NOTE: IN THE CURRENT VERSION OF DIOS, ONLY BRANCH 1 IS AL-
;       LOWED.  FURTHERMORE, ONLY THE SHORT FORM MAY BE SE-
;       LECTED.
;
* HOW MANY CRATES DO YOU WANT ON BRANCH 1? [1 TO 7] [N]:
```

Input the number of crates connected to branch 1. You must specify at least one and at most 7 crates.

```
; DEFINE BRANCH 1 CONTROLLER:

* INPUT CSR ADDRESS OF CONTROLLER <166000> [S]:
```

Input the octal Unibus address of the branch 1 CSR register, as wired in the Unibus interface. The address is truncated to 16 bits. The default value of 166000 is that given in Appendix D for branch 1.

```
* INPUT BASE ADDRESS OF CAMAC REGISTERS <164000> [S]:
```

Input the octal Unibus address of the base of the CAMAC register area, as wired in the branch 1 Unibus interface. The address is truncated to 16 bits. The default value of 164000 is that given in Appendix D for branch 1.

```
* INPUT BASE ADDRESS OF LAM VECTORS <400> [S]:
```

Input the base address of the LAM vector area, as wired in the branch 1 interrupt generator. The default of 400 is that given in appendix D for branch 1.

```
* INPUT INTERRUPT PRIORITY [4 TO 7] <4> [S]:
```

Input the priority as wired into the branch 1 interrupt generator. The default of 4 is that given in Appendix D for branch 1.

E.2.4.2 Defining a Borer Configuration -

The following questions are asked for each crate in a Borer system.

* INPUT BASE ADDRESS OF CAMAC REGISTERS <164000> [S]:
The address must be given as a 16-bit octal number, obtained by truncating the 18-bit Unibus address wired into the controller to the lower 16 bits. The default for the first crate is 164000 as shown; further crates have the defaults listed in Appendix D.

* INPUT BASE ADDRESS OF INTERRUPT VECTORS [OCTAL] <200> [S]:
Input the address as wired into the interrupt section of the crate controller. The address must be between 0 and 1600; the default, 200, is the same for all crates.

* WHAT IS THE INTERRUPT PRIORITY [4-7] <5> [S]:
Input the priority as wired in the interrupt section of the crate controller. Priorities 4, 5, 6, and 7 are allowed. The default is 5 for all crates.

* DO YOU WANT TO DEFINE THE NEXT CRATE? [Y/N]:
This question is not asked after the 10th crate has been defined. In this case, or if you answer no, the definition of the CAMAC configuration is completed. If you answer yes, logical crate n+1 is defined by repeating the questions in this section.

E.2.5 Specifying the Loader ACP

This section is skipped if you are not generating the loader ACP.

Otherwise, DIOSGEN executes the command file DD:[1,200] LDRACPGEN.CMD, which issues the questions and comments in this section. The results are used to create the TKB command file LDRACP.BLD, which is used to build LDRACP.TSK.

; GENERATE LOADER ACP:
;
* INPUT TASK PRIORITY [1 - 250(10), NO DECIMAL POINT] <240> [S]:
The loader ACP should normally operate at a higher priority than the driver ACP's or user tasks. If the default priority is inconvenient for some reason, another value may be selected. In this case, be sure a decimal number without a decimal point is input, as otherwise the task builder will fail. Normally you should type <CR> to select the default.

* DO YOU WANT A MAP? [Y/N]:
If you answer yes, LDRACP.MAP will be produced on the map device.

E.2.6 Specifying Driver ACP's

This section is skipped if no driver ACP's are being generated.

Otherwise, DIOSGEN executes the command file DD:[1,200]DRVACPGEN.CMD to carry out the driver ACP generation procedure as described in appendix F. The results of the questions are used to create the TKB command file DD:[1,200]DRVACP.BLD, which is used to build the driver ACP's in the DIOSGEN task-build phase.

E.2.7 Specifying the Message Output Processor

This section is skipped if the Message Output processor is not being generated.

Otherwise, DIOSGEN executes the command file DD:[1,200]MOACPGEN.CMD to issue the questions and comments described in this section. The results are used to create the TKB command file DD:[1,200]MOACP.BLD, which builds the Message Output driver and ACP.

```
; GENERATE THE MESSAGE OUTPUT PROCESSOR --
; GENERATE THE LOADABLE DRIVER FOR DEVICE MO0:
;
; THE DRIVER IS LOADED INTO A SYSTEM-CONTROLLED PARTITION.
* INPUT DRIVER PARTITION NAME [? IF UNKNOWN] <DRVPAR> [S]:
```

The driver partition may be any system-controlled partition. The default is DRVPAR. If you do not know the name of the partition, input a ? to obtain a list of the partitions; they are displayed exactly as previously described for the DAPAR partition, followed by these lines:

```
; NOTE THE NAMES OF THE SYSTEM-CONTROLLED PARTITIONS AND
; CHOOSE ONE FOR MESSAGE OUTPUT.  THE PARTITION "GEN" IS
; ALLOWED.
```

You are then again asked to input the partition name.

```
; GENERATE THE MESSAGE OUTPUT ACP:
;
; SPECIFY THE NUMBER OF LOGICAL UNITS NEEDED BY THE ACP.
; THIS NUMBER IS EQUAL TO THE TOTAL NUMBER OF TERMINALS
; (DEVICES OF TYPE TTN:) IN THE SYSTEM, PLUS 3.
; NOTE: THE NUMBER MUST BE DECIMAL, BUT MAY NOT HAVE
;       A DECIMAL POINT.
* INPUT THE NUMBER OF LOGICAL UNITS [? IF UNKNOWN] [S]:
```

The message output ACP assigns a separate LUN to each terminal in the system. In addition it needs a LUN for the Console Log device (CL0:), and for each of two message format files. If you know this number, type it in as a decimal integer without a decimal point.

If you do not know the number of terminals, input a "?" to obtain a list of the devices in the target system:

```
; HERE IS THE DEVICE LIST OF THE TARGET SYSTEM.
; COUNT THE NUMBER OF TERMINALS, INCLUDING THOSE MARKED
; OFFLINE, AND ADD 3.
INS [1,54]VMR
VMR @SS:[1,200]DEV
  (Display the devices with a VMR DEV command.)
        ...
TT0:
TT1:
TT2:    OFFLINE
        ...
TT13:
        ...
```

In this example there are 14 (octal!), or 12 (decimal) terminals; thus the number of units should be 12+3 or 15 . The number of units is again requested; in this example, you should type "15".

* INPUT TASK PRIORITY [1 - 250(10), NO DECIMAL POINT] <100> [S]:
  The task priority should be higher than that of most user tasks. If for some reason the given default value is inconvenient, select another value: it must be entered as a decimal integer with no decimal point. If an illegal priority is typed, TKB will fail when MOACP is built. Normally you should type <CR> to select the default of 100.

* DO YOU WANT A MAP? [Y/N]:
  This question is only asked if a map device was specified. If you answer yes, MODRV.MAP and MOACP.MAP will be produced when the MO components are built.

E.2.8 Specifying the DIOS Mount Processor

This section is skipped if the DIOS Mount processor is not being generated.

Otherwise, DIOSGEN.CMD executes DD:[1,200]DIOMOUGEN.CMD to issue the following questions and comments. The results are used to create DD:[1,200]DIOMOU.BLD, which is used to build DIOMOU.TSK.

```
; GENERATE THE DIOS MOUNT PROCESSOR:
;
```

* DO YOU WANT A MAP? [Y/N]:
  This question is not asked if no map device was specified. If you answer yes, DIOMOU.MAP is created on the map device.

E.2.9 Correcting Mistakes

All components of DIOS have now been specified.

The following files were created on DD:[1,200] --

```
DADRV.BLD;1     Build DADRV.TSK, DADRV.STB, and DIOLNK.TSK
LDRACP.BLD;1    Build LDRACP.TSK
DRVACP.BLD;1    Build driver ACP's (xxACP.TSK)
MOACP.BLD;1     Build MOACP.TSK and MODRV.TSK, MODRV.STB
DIOMOU.BLD;1    Build DIOMOU.TSK
```

The following file was created on TS:[1,54] --

```
DIODEF.CMD;1    Define DIOS parameters for command files.
```

One of the following macro files was created on DD:[1,220], depending on which CAMAC system was selected --

```
CAMDEF.ICP;1    Define ICP CAMAC configuration
CAMDEF.BOR;1    Define Borer CAMAC configuration
```

* DO YOU WISH TO CORRECT ANY MISTAKES AT THIS POINT? [Y/N]:
  If the user answers no, the rest of this section is skipped and the task-build phase is started.

* DO YOU WISH TO EDIT ANY FILES? [Y/N]:
  If you answer yes, ...AT. pauses to allow you to edit any incorrect files produced in the question phase:

```
; PLEASE EDIT INCORRECT FILES:
AT. -- PAUSING.  TO CONTINUE TYPE "RES ...AT."
```

You may now edit any incorrect files.

```
PIP @DD:[1,200]CLEANUP
```

After ...AT. is resumed, the edited files are purged and renamed to version 1.

* DO YOU WISH TO REPEAT ANY SPECIFICATIONS? [Y/N]:
  If you answer yes, the following list of questions appears:

```
; INDICATE WHICH COMPONENT SPECIFICATIONS YOU WISH TO REPEAT:
* SYSTEM AND CAMAC SPECIFICATIONS? [Y/N]:
* LOADER ACP SPECIFICATIONS? [Y/N]:
* DRIVER ACP SPECIFICATIONS? [Y/N]:
* MESSAGE OUTPUT PROCESSOR SPECIFICATIONS? [Y/N]:
```

Type "Y" after each component you wish to correct. The question phase will be repeated for the components indicated, if any.

E.2.10 Task-Build Phase

This section illustrates a complete DIOS generation, where all components were specified. Certain of the lines may not appear if a partial update of DIOS was performed.

```
; THE DIOS TASKS WILL NOW BE BUILT.
; THIS PHASE IS INDEPENDENT OF OPERATOR ACTION AND REQUIRES
; UP TO 15 MINUTES DEPENDING ON THE OPTIONS SELECTED.  WHEN
; THIS PHASE IS FINISHED, THE OPERATOR MUST CHECK THE RE-
; SULTS FOR ERRORS AND CONTINUE THE GENERATION PROCESS.
;
; BUILD THE DIOS MOUNT PROCESSOR:
TKB @DD:[1,200]DIOMOU.BLD
;
; TRANSFER DIOS SYSTEM MACROS AND OBJECT MODULES:
INS [1,54]LBR
LBR TS:[1,1]RSXMAC.SML/RP=DD:[1,220]DIOSMAC
LBR TS:[1,1]SYSLIB/RP=DD:[1,210]DIOSYM
REM ...LBR
```

After each /RP command, LBR will print a list of any modules replaced.

```
;
; TRANSFER COMMAND FILES:
;
PIP TS:[1,2]INSDIOS.CMD;*/DE
PIP TS:[1,2]INSDIOS.CMD;1=DD:[1,200]INSDIOS.CMD
;

; ASSEMBLE CAMAC DEFINITION FILE:
MAC DD:[1,210]CAMDEF;1=DD:[1,220]CAMDEF.BOR/PA:1,CAMTBL.BOR
MAC DD:[1,210]CAMDEF;1=DD:[1,220]CAMDEF.ICP/PA:1,CAMTBL.ICP
```

The above 3 lines do not appear if there is no CAMAC system; otherwise only the MAC command line corresponding to the system type being generated appears.

```
;
; BUILD SYSTEM-RESIDENT ROUTINES:
TKB @DD:[1,200]DADRV.BLD
;
; BUILD THE LOADER ACP:
TKB @DD:[1,200]LDRACP.BLD
;
; BUILD THE DRIVER ACP'S:
TKB @DD:[1,200]DRVACP.BLD
;
; BUILD THE MESSAGE OUTPUT PROCESSOR:
TKB @DD:[1,200]MOACP.BLD
;
; TRANSFER THE MESSAGE FILES:
PIP TS:[1,2]*.MSG;1=DD:[1,210]*.MSG
PIP TS:[1,2]*.MSG/PU
;
; END OF TASK BUILD PHASE.
```

E.2.11 Correcting Errors from Task-Build Phase

* DID ANY ERRORS OCCUR DURING ASSEMBLY OR TASK BUILDING? [Y/N]:
Answer yes if any of the MCR commands to MAC, PIP, TKB, or LBR produced any error messages. If no errors occurred, the rest of this section is skipped.
If yes:

```
; PLEASE CORRECT ANY ERRORS AT THIS TIME.
; EDIT ANY INCORRECT COMMAND OR MACRO-11 FILES AND REPEAT
; THE STEP WHICH CAUSED THE ERROR BY MANUALLY TYPING THE
; COMMANDS LISTED WITH THIS STEP.
AT. -- PAUSING.  TO CONTINUE TYPE "RES ...AT."
```

Determine from the listed error messages from PIP, LBR, TKB, or MAC what errors occurred and how to correct them. Take any measures necessary to correct the errors, such as editing command files or making room on disks. Then repeat the steps which caused the errors by manually typing in the associated MCR command.

If necessary, DIOSGEN may be aborted at this point and restarted by typing

```
ABO ...AT.
@DD:[1,200]DIOSGEN
```

This time, answer "yes" when asked if a DIOS system has already been generated. Avoid respecifying any components which were generated successfully.

E.2.12 Installing the DIOS System

```
; ALL TASKS NEEDED FOR THE DIOS SYSTEM ARE NOW READY.
* DO YOU WISH TO INSTALL DIOS AT THIS TIME? [Y/N]:
```

If you answer no, DIOSGEN exits after typing the following instructions:

```
; TO COMPLETE INSTALLING THE DIOS SYSTEM, BOOT THE TARGET
; SYSTEM AND TYPE THE INDIRECT COMMAND:
;
; @[1,2]INSDIOS.CMD
;
; THE INDIRECT FILE WILL GUIDE YOU THROUGH THE INSTALLATION.
@<EOF>
```

You must then complete the DIOS generation as indicated.

* IS THE TARGET SYSTEM RUNNING AND ASSIGNED TO SY0? [Y/N]:
Answer no if you are generating DIOS for a system other than the one you are running under. If you answer no, the above instructions for completing the DIOS generation are printed, and DIOSGEN ends execution.

If you answer yes, DIOSGEN executes [1,2]INSDIOS.CMD directly, which carries out the installation of DIOS as described

below. The same procedure is followed as if INSDIOS had been executed manually.

```
; INSTALL THE DIOS SYSTEM:
;
; THE RSX MOUNT PROCESSOR MUST BE REPLACED WITH THE DIOS-
; COMPATIBLE MOUNT PROCESSOR WHICH IS CAPABLE OF MOUNTING
; DA: AND MO:.
;
* HAVE YOU ALREADY REPLACED THE RSX MOUNT PROCESSOR? [Y/N]:
```

You must answer no unless an existing DIOS system is being updated and the DIOS mount processor was not modified. If you answer no, the mount processor is replaced as follows:

```
INS [1,54]VMR
VMR @[1,2]RPLMOU
REM ...VMR
REM ...MOU
INS [1,54]DIOMOU
PIP [1,2]RPLMOU.CMD;*/DE
```

You are now asked to specify which driver ACP's are to be installed with DIOS. Using this information, INSDIOS creates a new version of INSDIOS.CMD which performs the actual installation and activation of DIOS.

```
; INPUT THE SET OF DRIVER ACP'S TO BE INSTALLED WITH DIOS:
; FOR EACH ACP, TYPE ITS 2-CHARACTER IDENTIFIER.
; TYPE <CR> WHEN ALL THE ACP'S HAVE BEEN SPECIFIED.
;
* INPUT NEXT ACP ID [2 CHARS.] [S]:
```

Type the driver ACP id or <CR> as indicated. The question will be repeated until <CR> is typed. For each ACP a command of the form INS [1,54]xxACP is included in the new version of INSDIOS being created.

```
; INSTALL AND ACTIVATE THE DIOS SYSTEM:
;
```

The new version of INSDIOS.CMD is now executed to install DIOS for the first time.

```
INS [1,54]LOA
LOA MO:
REM ...LOA
INS [1,54]MOACP
MOU MO:
```

The above five lines load the message output driver, install the message output ACP, and mount MO0: to activate the message output processor.

```
INS [1,54]DADRV
SET /NOMAIN=DAPAR
SET /MAIN=DAPAR:xxxx:15:TASK
```

xxxx gives the base of the DAPAR partition.

```
RUN [1,54]DIOLNK
INS [1,54]LDRACP
INS [1,54]xxACP
  ...
(Install further driver ACP's)
  ...
MOU DA:
```

The last group of commands installs the basic DIOS system. First, the resident routines are installed into partition DAPAR; DAPAR is then converted to a task partition in order to prevent accidentally reinstalling DADRV. Then DIOLNK is run, linking the DA0: device tables to the end of the RSX device list. The loader and driver ACP's are installed next. Finally DA0: is mounted, activating the DIOS system. The new DIOSGEN file then exits, returning to the old version.

```
;
; THE DIOS SYSTEM IS NOW INSTALLED AND ACTIVATED.
;
; IF DESIRED, DIOS MAY BE PERMANENTLY INSTALLED BY SAVING
; THE RSX SYSTEM IN THE CURRENT STATE WITH THE SAV COMMAND
; AFTER ALL DEVICES HAVE BEEN DISMOUNTED.
; TO ACTIVATE DIOS YOU MUST THEN SIMPLY EXECUTE
;
; MOU MO: -- ACTIVATE MESSAGE OUTPUT PROCESSOR
; MOU DA: -- ACTIVATE DIOS I/O
;
; WHENEVER DIOS IS NEEDED.
;
; ALTERNATIVELY, YOU MAY INCLUDE THE COMMAND
;
; @[1,2]INSDIOS
;
; IN THE SYSTEM STARTUP FILE.  THE DIOS SYSTEM WILL THEN
; BE INSTALLED AND ACTIVATED EACH TIME THE SYSTEM IS BOOTED.
PIP [1,2]INSDIOS.CMD/PU
@ <EOF>
```

INSDIOS is now finished; before exiting, the old version is deleted.

If you choose to install DIOS permanently, you must first make a backup copy of RSX11M.SYS, or be sure that RSX11M.TSK is available. Since it is impossible to remove a permanently installed DIOS system, any modifications in the resident portion of DIOS (changes in the CAMAC configuration, for example) require restoring the original RSX system.

When the backup has been made, execute the following commands:

```
INS [1,54]SAV
DMO DA:
DMO MO:
  ...
(Dismount all other mounted devices)
  ...
DMO DK0:
SAV
```

The saved system will now be rebooted.

If you choose not to install DIOS permanently, you must re-install it each time the RSX system is booted. This is done by executing the new verion of INSDIOS.CMD when DIOS is needed; you may include a command to execute INSDIOS in the system startup file so that this will be done automatically.

** WARNING **

If you need to save the system for any reason, be sure that DIOS is not installed; once DIOS is saved it cannot be removed, as mentioned above.

After generating DIOS, the system disk may be cleaned up as follows:

* Delete [1,54]MOU.TSK if present.

* Purge all files on [1,54]. This deletes, in particular, old versions of DIOMOU.TSK.

** WARNING **

Do not perform the above two steps if the Mount processor was not successfully replaced! Doing so will result in an unusable system.

* If DIOS is permanently installed, you may delete [1,54]DADRV.TSK, [1,2]INSDIOS.CMD and [1,54]DIOLNK.TSK. Do not delete these files if you intend to reinstall DIOS after booting.

APPENDIX F

Generating Driver ACP's

From time to time the user of DIOS may wish to add new module types to his configuration, or to stop using old, obsolete types. In this case, he must rebuild his driver ACP's, adding or removing module drivers as needed. Other considerations, such as grouping modules used at different times into different ACP's to decrease the size of the largest ACP, may also warrant rebuilding the ACP's. This appendix describes how to generate driver ACP's using command files on the DIOS distribution medium.

F.1 Task Images

Driver ACP task images are kept on the DIOS system disk under UIC [1,54]. The general form of the filename is xxACP.TSK, where xx stands for the 2-character identifier of the driver ACP. The purpose of the driver ACP generation is to specify and build the task images for one or more of these ACP's.

F.2 Input Files

Most files necessary for generating driver ACP's are on the medium on which DIOS is distributed. Other files needed are created on the system disk by the DIOS and RSX-11M generation processes. The relevant files are the following:

Distribution Device:

[1,200]DRVACPGEN.CMD	Driver ACP generation command file
[1,210]DRVACP.OLB	Main driver ACP routines
	Distributed DIOS module drivers

System Device:

[1,54]DADRV.STB	DIOS resident routine addresses
[1,54]RSX11M.STB	RSX system routine addresses
[1,54]DIODEF.CMD	Define DIOS parameters to DRVACPGEN

F.2.1 User-Written Drivers

The user may wish to include drivers for non-standard modules which are not distributed in DRVACP.OLB. If so, he must first create the object code for the driver according to the specifications given in "Guide to Writing a DIOS I/O Driver". He then has two options:

1. He may insert the driver object code into DRVACP.OLB using LBR, and specify the driver as if it were distributed with DIOS. This is the simplest method.

2. If desired, the object code may be kept in another file which may be specified separately when the driver ACP is generated as discussed below.

F.3 Preparing for ACP Generation

The preparations for generating driver ACP's are similar to those for generating the DIOS system.

* Determine each ACP identifier to be assigned to the new driver ACP, and decide which drivers it is to contain.

* Mount the target system disk if it is not already running. Make sure it is write-enabled.

* Mount a copy of the DIOS distribution medium. Make sure it is write-enabled.

* Make sure the object files for any user-written drivers are on an accessible medium.

* Log on under a privileged account and set the UIC to [1,200].

* Initiate the driver ACP generation process by typing

```
@ddn:DRVACPGEN
```

where ddn stands for the distribution device name. The command file then issues the questions and comments discussed in the next section.

F.4 Details of Generation Procedure

DRVACPGEN.CMD operates in two phases:

In the dialog phase, the characteristics of the ACP's to be generated are specified by answering a series of questions asked by the command file. The results are used to create [1,200]DRVACP.BLD on the distribution device.

In the task-build phase, DRVACP.BLD is submitted as an indirect command file to TKB, which then builds all the ACP tasks specified.

DRVACPGEN may also be executed during the main DIOS generation process, in which case certain questions are omitted as they have previously been answered. The differences are indicated below when appropriate.

F.4.1 Assigning Devices

This section is skipped if Driver ACP's are being generated as part of an overall DIOS generation. Otherwise, introductory comments and questions are printed and the devices are assigned almost exactly as in the setup section of Appendix E. The devices are assigned as follows:

Distribution Device = DD:
Target System Device = TS:
Map Device = MP: if specified.

F.4.2 Specifying a Driver ACP

More than one driver ACP may be specified. If so, the questions in this section are repeated for each ACP to be built.

```
; GENERATE DIOS DRIVER ACP'S:
;
; SPECIFY THE NEXT DRIVER ACP:
* INPUT THE 2-CHARACTER ACP IDENTIFIER [S]:
  Type "xx" to generate [1,54]xxACP.TSK.

* DO YOU WISH TO DELETE OLDER VERSIONS OF xxACP? [Y/N]:
```

Answer yes if older versions of the ACP exist under [1,54] of the DIOS system disk and you no longer wish to keep them. Answer no if no such files exist, or if you do not want to delete them.

```
PIP TS:[1,54]xxACP.TSK;*/DE
```

The old task images are deleted if so specified.

* INPUT THE TASK PRIORITY [1-250(10), NO DECIMAL POINT] <240> [S]:
The driver ACP should have a task priority higher than that of any user task, but lower than the loader ACP. If the given default is inconvenient for some reason, you may select another priority. If you do, the priority given must be a valid argument of the PRI= Task Builder option. If it is not, TKB will fail when the ACP is built. Normally you should type <CR> to select the default of 240.

* DO YOU WANT A MAP? [Y/N]:
This question is asked only if a map device was specified. If you answer yes, xxACP.MAP will be created for this ACP.

F.4.3 Specifying Module Drivers to Include

```
; SPECIFY THE MODULE DRIVERS TO BE INCLUDED:
;
; FIRST SPECIFY THE TOTAL NUMBER OF DRIVERS TO INCLUDE.
; NOTE THAT THIS INCLUDES BOTH DISTRIBUTED AND USER-WRITTEN
; DRIVERS.
;
; THEN SPECIFY ALL THE DRIVERS FROM THE DISTRIBUTED OBJECT
; LIBRARY.  THIS IS DONE BY ENTERING THE TYPE CODE OF EACH
; MODULE WHOSE DRIVER IS TO BE INCLUDED, AS REQUESTED.
;
; IF USER-WRITTEN DRIVERS NOT CONTAINED IN THE DISTRIBUTED
; LIBRARY ARE TO BE INCLUDED, TYPE <CR> INSTEAD OF A MODULE
; TYPE CODE WHEN ALL DRIVERS FROM THE LIBRARY HAVE BEEN SPE-
; CIFIED.  YOU WILL LATER BE ASKED TO EDIT THE TKB COMMAND
; FILE TO INCLUDE THE ACTUAL OBJECT FILE SPECIFICATIONS.
;
```

* INPUT THE TOTAL NUMBER OF DRIVERS [OCTAL] [S]:
The number given must be a valid argument of the GBLDEF=$DRVRS:nnnnnn task-builder option. If it is invalid, TKB will fail when this ACP is built.

* WHAT IS THE NEXT MODULE TYPE? [S]:
To include module xxDRV from the distributed library, type "xx". This question will be repeated until the total number of drivers requested is specified, or <CR> is typed.

If you have specified all of the drivers from the library you wish to include, type <CR>. A line containing "<USER-WRITTEN DRIVER>" is inserted into the TKB command file being created for each driver remaining to be specified. These lines must later be replaced by valid object file specifications.

When all drivers have been specified, the following question is asked:

* DO YOU WANT TO BUILD ANY MORE DRIVER ACP'S [Y/N]:
If you answer yes, the preceding questions are repeated to define the next driver ACP. If you answer no, the next sec-

tion is executed.

F.4.4 Including User-Written Drivers

If you did not specify any drivers not in the library, this section is skipped.

```
; EDIT TASK BUILD FILE TO INCLUDE USER-WRITTEN DRIVERS:
; REPLACE LINES CONTAINING "<USER-WRITTEN DRIVER>"
; WITH OBJECT FILE SPECIFICATION.
EDI DD:[1,200]DRVACP.BLD
```

EDI is now entered with the TKB command file. Replace the lines as indicated above with all object file specifications needed for the non-library drivers. You must specify as many drivers as there are lines to replace. When done, exit from the editor with an "ED" command; the ACP's will then be built.

F.4.5 Building the Driver ACP's

This section is skipped if driver ACP's are being generated as part of an overall DIOS generation; in this case, the building of the ACP's is postponed until the DIOSGEN task-build phase.

If DRVACPGEN alone is executed, the ACP's are now built as follows:

```
; BUILD DRIVER ACP'S:
TKB @DD:[1,200]DRVACP.BLD
```

If any TKB errors occurred, you may either edit DRVACP.BLD to correct the errors and manually re-issue the TKB command line above, or you may simply repeat the ACP generation to redefine any ACP's which were specified incorrectly.

After TKB is finished, the command file exits with the following message:

```
; THE SPECIFIED DRIVER ACP'S HAVE NOW BEEN BUILT.
; PLEASE EDIT [1,2]INSDIOS.CMD TO INSTALL THE NEW ACP'S,
; AND REMOVE COMMANDS TO INSTALL ACP'S WHICH ARE NO LONGER
; NEEDED.  ALSO DELETE THE TASK IMAGES OF ANY DRIVER ACP'S
; WHICH ARE NO LONGER BEING USED.
@ <EOF>
```

You should follow the printed suggestions to complete the incorporation of the new driver ACP's into the DIOS system.

F.5 Task Build Command File

The following is a sample task-build file for a typical driver ACP. The comments have been added to illustrate the function of the various sections. Note that one user-written driver specification must be edited.

```
; BUILD DRIVER ACP WITH ACP NAME "PD".
; OUTPUT FILES:
;       TASK IMAGE = PDACP.TSK.  SPECIFY ACP WITH /AC SWITCH
;       MAP FILE = PDACP.MAP
; INPUT FILES:
;       OBJECT LIBRARY = DRVACP.OLB
;
; SPECIFY MAIN DRIVER ACP CONTROLLING ROUTINES:
;
TS:[1,54]PDACP/AC,MP:PDACP=DD:[1,210]DRVACP/LB:DRVACP:$RQDSP
;
; SPECIFY MODULE DRIVERS FROM DISTRIBUTED OBJECT LIBRARY:
;
DD:[1,210]DRVACP/LB:FKDRV:TGDRV:TSDRV:MXDRV:CMDRV
;
; SPECIFY USER-WRITTEN DRIVERS:
;
DK2:[44,30]MADRV.OBJ     ;OBJECT FILE FOR U.W.D "MA"
<USER-WRITTEN DRIVER>    ;REPLACE THIS LINE WITH VALID
                         ;DRIVER OBJECT FILE SPECIFICATION
                         ;AS IN PREVIOUS LINE
;
; SPECIFY SYMBOL TABLES AND LIBRARIES:
;
DD:[1,210]DRVACP/LB
TS:[1,1]DAPAR.STB/SS,[1,54]RSX11M.STB/SS
/
; SPECIFY OPTIONS:
;
TASK=PD....      ;TASK NAME = PD....
PRI=240          ;PRIORITY (HIGHER THAN MOST USER TASKS)
ASG=MO:3         ;ASSIGN LUN FOR ERROR MESSAGE OUTPUT
STACK=40         ;SMALLEST STACK POSSIBLE
GBLDEF=$DRVRS:7 ;TOTAL OF 7 DRIVERS IN ACP
//
```

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