RSX–11M–PLUS and Micro/RSX I/O Drivers Reference Manual

Order No. AA–JS11A–TC

RSX–11M–PLUS Version 4.0
Micro/RSX Version 4.0

Digital Equipment Corporation Maynard, Massachusetts

First Printing, May 1979
Revised, December 1981
Revised, July 1985
Revised, September 1987

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	EduSystem	UNIBUS
DEC/CMS	IAS	VAX
DEC/MMS	MASSBUS	VAXcluster
DECnet	MicroPDP–11	VMS
DECsystem–10	Micro/RSX	VT
DECSYSTEM–20	PDP	
DECUS	PDT	
DECwriter	RSTS	digital
DIBOL	RSX	

ZK3078

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*	CANADA	INTERNATIONAL
Digital Equipment Corporation	Digital Equipment of Canada Ltd.	Digital Equipment Corporation PSG Business Manager
P.O. Box CS2008	100 Herzberg Road	c/o Digital's local subsidiary
Nashua, New Hampshire 03061	Kanata, Ontario K2K 2A6	or approved distributor
	Attn: Direct Order Desk	

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by TEX, the typesetting system developed by Donald E. Knuth at Stanford University. TEX is a trademark of the American Mathematical Society.

Contents

Chapter 2 Full-Duplex Terminal Driver

Chapter 3 Virtual Terminal Driver

Chapter 4 Disk Drivers

Chapter 5 DECtape II Driver

Chapter 6 Magnetic Tape Drivers

Chapter 7 Line Printer Driver

Chapter 8 Null Device Driver

Part II: RSX–11M–PLUS Drivers

Chapter 9 Card Reader Driver

Chapter 10 QIO DEUNA Driver

Chapter 11 PCL11 Parallel Communications Link Drivers

Chapter 12 Laboratory Peripheral Accelerator Driver

Chapter 13 K-Series Peripheral Support Routines

Chapter 14 UNIBUS Switch Driver

Appendix A Summary of I/O Functions

Appendix B I/O Function and Status Codes

Index

Figures

Tables

Preface

Manual Objectives

This manual provides all the information needed to interface directly with the I/O device drivers supplied as part of the RSX-11M-PLUS and Micro/RSX operating systems.

Intended Audience

This manual is for experienced RSX-11M-PLUS and Micro/RSX programmers who want to take advantage of the time and/or space savings that result from direct use of the I/O drivers. Readers should be familiar with the information contained in the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual*, should have some experience using the Task Builder (TKB) and either MACRO-11 or FORTRAN programs, and should be familiar with the manuals describing their use.

Structure of This Document

Chapter 1 provides an overview of RSX-11M-PLUS and Micro/RSX input/output operations. It introduces you to logical unit numbers (LUNs), Directive Parameter Blocks (DPBs), event flags, macro calls, and so on; includes discussions of the standard I/O functions common to a variety of devices; and summarizes standard error and status conditions relating to completion of I/O requests.

Chapters 2 to 14 describe the use of all device drivers supported by RSX-11M-PLUS and Micro/RSX. Each of these chapters is structured in similar fashion and focuses on the following basic elements:

- The device, including information on physical characteristics such as speed, capacity, access, and usage
- The standard functions that the devices support and descriptions of device-specific functions
- The special characters, carriage control codes, and functional characteristics
- The error and status conditions that the driver returns on acceptance or rejection of I/O requests
- Programming hints

Appendix A provides quick reference material on I/O functions. Refer to the *RSX-11M-PLUS and Micro/RSX I/O Operations Reference Manual* for more information on status codes.

Appendix B lists numeric codes for all I/O functions, directive status, returns and I/O completion status returns.

Appendix C includes the source code for I/O error codes, Directive Status Word (DSW) error codes, and I/O function codes.

Associated Documents

The following RSX-11M-PLUS and Micro/RSX manuals may be useful:

- *RSX-11M-PLUS Information Directory and Master Index*
- *RSX-11M-PLUS and Micro/RSX Executive Reference Manual*
- *RSX-11M-PLUS and Micro/RSX Task Builder Manual*
- *PDP-11 MACRO-11 Language Reference Manual*
- *RSX-11M-PLUS Release Notes*
- *Micro/RSX Base Kit Release Notes*
- *Micro/RSX Release Notes*

In addition, documentation for programming in any of the PDP-11 languages may be helpful.

Conventions Used in This Document

The following conventions are observed in this manual:

Convention	Meaning
>	A right angle bracket is the default prompt for the Monitor Console Routine (MCR), which is one of the command interfaces used on RSX–11M–PLUS and Micro/RSX systems.
$	A dollar sign followed by a space is the default prompt of the DIGITAL Command Language (DCL), which is one of the command interfaces used on RSX–11M–PLUS and Micro/RSX systems. Many systems include DCL.
MCR>	This is the explicit prompt of the Monitor Console Routine (MCR).
DCL>	This is the explicit prompt of the DIGITAL Command Language (DCL).
UPPERCASE	Uppercase letters in a command line indicate letters that must be entered as they are shown. For example, utility switches must always be entered as they are shown in format specifications.
command abbreviations	Where short forms of commands are allowed, the shortest form acceptable is represented by uppercase letters. The following example shows the minimum abbreviation allowed for the DCL command DIRECTORY: `$ DIR`
lowercase	Any command in lowercase must be substituted for. Usually the lowercase word identifies the kind of substitution expected, such as a filespec, which indicates that you should fill in a file specification. For example: `filename.filetype;version` This command indicates the values that comprise a file specification; values are substituted for each of these variables as appropriate.
/keyword, /qualifier, or /switch	A command element preceded by a slash (/) is an MCR keyword; a DCL qualifier; or a task, utility, or program switch. Keywords, qualifiers, and switches alter the action of the command they follow.
parameter	Required command fields are generally called parameters. The most common parameters are file specifications.
[option]	Square brackets indicate optional entries in a command line or a file specification. If the brackets include syntactical elements, such as periods (.) or slashes (/), those elements are required for the field. If the field appears in lowercase, you are to substitute a valid command element if you include the field. Note that when an option is entered, the brackets are not included in the command line.
[,...]	Square brackets around a comma and an ellipsis mark indicate that you can use a series of optional elements separated by commas. For example, (argument [,...]) means that you can specify a series of optional arguments by enclosing the arguments in parentheses and by separating them with commas.

Convention	Meaning
{ }	Braces indicate a choice of required options. You are to choose from one of the options listed.
:argument	Some parameters and qualifiers can be altered by the inclusion of arguments preceded by a colon. An argument can be either numerical (COPIES:3) or alphabetical (NAME:QIX). In DCL, the equal sign (=) can be substituted for the colon to introduce arguments. COPIES=3 and COPIES:3 are the same.
( )	Parentheses are used to enclose more than one argument in a command line. For example: `SET PROT = (S:RWED,O:RWED)` Commas are used as separators for command line parameters and to indicate positional entries on a command line. Positional entries are those elements that must be in a certain place in the command line. Although you might omit elements that come before the desired element, the commas that separate them must still be included.
[g,m] [directory]	The convention [g,m] signifies a User Identification Code (UIC). The g is a group number and the m is a member number. The UIC identifies a user and is used mainly for controlling access to files and privileged system functions. This may also signify a User File Directory (UFD), commonly called a directory. A directory is the location of files. Other notations for directories are: [ggg,mmm], [gggmmm], [ufd], [name], and [directory]. The convention [directory] signifies a directory. Most directories have 1- to 9-character names, but some are in the same [g,m] form as the UIC. Where a UIC, UFD, or directory is required, only one set of brackets is shown (for example, [g,m]). Where the UIC, UFD, or directory is optional, two sets of brackets are shown (for example, [[g,m]]).
filespec	A full file specification includes device, directory, file name, file type, and version number, as shown in the following example: `DL2:[46,63]INDIRECT.TXT;3` Full file specifications are rarely needed. If you do not provide a version number, the highest numbered version is used. If you do not provide a directory, the default directory is used. Some system functions default to particular file types. Many commands accept a wildcard character (*) in place of the file name, file type, or version number. Some commands accept a filespec with a DECnet node name. A period in a file specification separates the file name and file type. When the file type is not specified, the period may be omitted from the file specification.

Convention	Meaning
;	A semicolon in a file specification separates the file type from the file version. If the version is not specified, the semicolon may be omitted from the file specification.
@	The at sign invokes an indirect command file. The at sign immediately precedes the file specification for the indirect command file, as follows: `@filename[.filetype;version]`
...	A horizontal ellipsis indicates the following: • Additional, optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis shows where elements of command input or statements in an example or figure have been omitted because they are irrelevant to the point being discussed.
KEYNAME	This typeface denotes one of the keys on the terminal keyboard, for example, the RETURN key.
CTRL/a	The symbol CTRL/a means that you are to press the key marked CTRL while pressing another key. Thus, CTRL/Z indicates that you are to press the CTRL key and the Z key together in this fashion. CTRL/Z is echoed on some terminals as ^Z. However, not all control characters echo.
n	A lowercase n indicates a variable for a number.
xxx	A symbol with a 1- to 3-character abbreviation, such as x or RET, indicates that you press a key on the terminal. For example, RET indicates the RETURN key, LF indicates the LINE FEED key, and DEL indicates the DELETE key.

In addition, unless otherwise noted, the term "RSX–11" refers to both the RSX–11M–PLUS and Micro/RSX operating systems.

Note that while RSX–11M–PLUS and Micro/RSX systems require certain parameters, they ignore them. These parameters are necessary to maintain compatibility with RSX–11D.

Furthermore, except in MACRO–11 coding examples, all numbers are assumed to be decimal unless otherwise specified. In MACRO–11 coding examples, the reverse is true: all numbers are considered to be octal unless followed by a decimal point (which indicates a decimal number).

Finally, in FORTRAN subroutine models, parameters that begin with the letters i to n indicate integer variables. In general, where a call uses both i and n prefixes, the i form indicates the name of an array and the n form specifies the size of the array.

All integer arrays and variables are assumed to occupy one storage word for each variable (that is, INTEGER*2) and all real arrays and variables are assumed to occupy two storage words for each variable (that is, REAL*4).

Summary of Technical Changes

The following sections list features, commands, qualifiers, error messages, and restrictions that are new to I/O drivers for the RSX–11M–PLUS and Micro/RSX Version 4.0 operating systems. These new or modified features are documented in this revision of the *RSX–11M–PLUS and Micro/RSX I/O Drivers Reference Manual.*

Also, major changes to the organization of the manual are included at the end of this summary.

New Hardware Support

RSX–11M–PLUS and Micro/RSX Version 4.0 support the following new hardware:

- The LQP03 letter-quality printer
- The LA75 dot matrix printer and LCG01 graphics printer
- The LA210 and LA2XX-series personal printers
- The DHU11 and DHQ11 asynchronous multiplexers
- The CXA16, CXB16, and CXY08 BA200-series multiplexers
- The RA82, RD32, and RD54 Winchester disk drives
- The RX33 single flexible disk

New Features

The terminal driver (TTDRV) has the following new features:

- Support for the following terminal characteristics:

Characteristic	Function
TC.CLN	7- or 8-bit character size at hardware level
TC.SXL	Printer supports sixel graphics
TC.MAP	Local Area Terminal (LAT) mapping
TC.QDP	Connect/disconnect/queue-depth of LAT application terminal

- A new hardware bit characteristic (7- and/or 8-bit characteristic) that supports nonterminal devices. This characteristic, which is defined in the TTSYM module, determines that the eighth bit is not required if the 7-bit characteristic is selected. The characteristic can be set using either the Monitor Console Routine (MCR) command SET /CHAR_LENGTH or the Digital Command Language (DCL) command SET TERMINAL/CHARACTER_LENGTH.
- Nonprivileged tasks can now issue the subfunction breakthrough write (TF.WBT) to the tasks' terminal. However, nonprivileged tasks cannot issue TF.WBT to other terminals.
- To improve task performance, TTDRV no longer uses intermediate buffers when performing Q-bus direct memory access (DMA) using the instruction write logical block and pass all characters (IO.WAL).
- TTDRV no longer requires that a terminal be attached for asynchronous system trap (AST) notification (IO.ATA) to set the terminal characteristics TC.OOB, TC.ICS, and TC.SCA.

Changes to the Document

The following changes in organization are included in this revision of the *RSX-11M-PLUS and Micro/RSX I/O Drivers Reference Manual:*

- The manual is now divided into the following parts:
  - Part 1: Common Drivers

    This part of the manual is devoted to I/O drivers that are common to both RSX-11M-PLUS and Micro/RSX.
  - Part 2: RSX-11M-PLUS Drivers

    This part of the manual is devoted to I/O drivers that are available only to RSX-11M-PLUS.
- A new appendix (Appendix C) that includes the source code for I/O error codes, Directive Status Word (DSW) error codes, and I/O function codes has been added.
- The *RSX-11M-PLUS and Micro/RSX I/O Drivers Reference Manual* appendix that included information about the QIO$ interface to the Ancillary Control Processors (ACPs) is now in the *RSX-11M-PLUS and Micro/RSX I/O Operations Reference Manual.*

Part I: Common Drivers

Chapter 1
Operating System Input/Output

1.1 Overview of RSX-11M-PLUS and Micro/RSX I/O

The RSX-11M-PLUS and Micro/RSX operating systems support a wide variety of input and output devices, including disks, DECtapes, magnetic tapes, tape cassettes, line printers, card readers, and such laboratory and industrial devices as analog-to-digital (A/D) converters, universal digital controllers, and laboratory peripheral systems.

Digital Equipment Corporation supplies the drivers for these devices as part of the system software. This manual describes all the device drivers that the RSX operating system supports and the characteristics, functions, error conditions, and programming hints associated with each. You can add devices that this manual does not describe to basic RSX system configurations, but you must develop and maintain your own drivers for these devices. (See the *RSX-11M-PLUS and Micro/RSX Guide to Writing an I/O Driver.*)

Input/output operations under RSX-11M-PLUS and Micro/RSX are extremely flexible and are as device- and function-independent as possible. Programs issue I/O requests to logical units that you previously associated with particular physical device units. Each program or task can establish its own correspondence between physical device units and logical unit numbers (LUNs). The Executive queues I/O requests as your task issues them and subsequently processes them according to the relative priority of the tasks that issued them. Your tasks can issue I/O requests for appropriate devices through either the File Control Services (FCS) or Record Management Services (RMS), or your tasks can interface directly to an I/O driver by the Queue I/O (QIO$) Executive directive macro.

Your task requests all of the I/O services that this manual describes by using QIO$ Executive directive macros. A function code that you include in the QIO$ macro indicates the particular input or output operation that the system and the driver are to perform. I/O functions can request operations such as the following:

- Attaching or detaching a physical device unit for a task's exclusive use
- Reading or writing a logical or virtual block of data
- Canceling a task's I/O requests

QIO macros can also specify a wide variety of device-specific I/O operations (for example, reading DECtape in reverse and rewinding cassette tape).

1.2 Physical, Logical, and Virtual I/O

An I/O transfer can take place in three possible modes: physical, logical, and virtual. A description of each mode follows:

I/O Mode	Description
Physical	Takes place by reading and writing data in the actual physical units that the hardware accepts (for example, sectors on a disk). For most devices, physical I/O is identical to logical I/O. For example, the RK05 cartridge disk has sectors of 256 words, the same size as RSX–11M–PLUS and Micro/RSX logical blocks for all disks. Thus, for the RK05, a logical block maps directly into a physical block. However, the mapping is not one to one for other devices. The RF11 fixed-head disk, for example, is word addressable, but no physical I/O may be done with the RF11. Data is always written in 256-word logical blocks. The system records data for the RX01 flexible disk in physical sectors of 64 words each. Therefore, logical blocks for the RX01 are made up of four physical sectors.
Logical	Takes place by reading and writing data in blocks convenient for the operating system. For most devices logical blocks map directly into physical blocks. For block-structured devices (for example, disks), logical blocks are numbered, beginning with 0. For non-block-structured devices (for example, terminals), logical blocks are not addressable.
Virtual	Takes place by reading and writing data to open files. In this case, the Executive maps virtual blocks into logical blocks. For file-structured devices (disks or DECtapes), virtual blocks are the same size as logical blocks, are numbered starting from 1, and are relative to the file rather than to the device. For non-file-structured devices, the mapping from virtual block to logical block is direct.

1.3 Logical Units

This section describes the construction of the Logical Unit Table (LUT) and the use of logical unit numbers (LUNs).

1.3.1 Logical Unit Number

A logical unit number, or LUN, is a number that the system associates with a physical device unit during RSX–11M–PLUS and Micro/RSX I/O operations. For example, you might associate LUN 1 with one of the terminals in the system; LUNs 2, 3, 4, and 5 with DECtape drives; and LUNs 6, 7, and 8 with disk units. The association is a dynamic one; each task running in the system can establish its own correspondence between LUNs and physical device units, and the system can change that association at almost any time. This dynamic, flexible association is a major factor in the device-independent programming of the system.

A LUN is simply a short name for the association between a logical unit and a physical device unit. Once the association has been made, the LUN provides a direct and efficient mapping to the physical device unit, thus eliminating the searching of device tables whenever the system encounters a reference to a physical device unit.

Remember that, although you or a task can change the association of a LUN to a physical device unit at any time, reassigning a LUN at run time causes pending I/O requests for the previous LUN assignment to be canceled. Therefore, you must verify that all outstanding I/O requests for a LUN have been serviced before you associate that LUN with another physical device unit.

1.3.2 Logical Unit Table

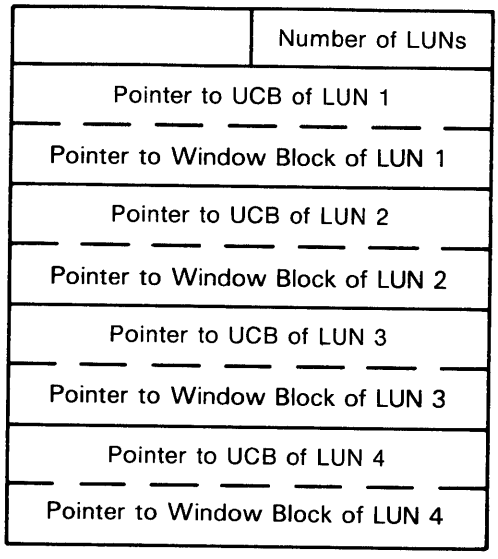
There is one Logical Unit Table (LUT) for each task running in the operating systems. The task header contains this table as a variable-length block. Each LUT contains enough 2-word entries for the number of logical units. You specify the number of logical units in the Task Builder (TKB) by the "UNITS=" option when you build your task.

The first word of each 2-word entry contains a pointer to the Unit Control Block (UCB) that represents the physical device unit currently associated with that LUN. This linkage may be indirect; that is, you may force redirection of references from one unit to another unit with the Digital Command Language (DCL) command ASSIGN/REDIRECT. The second word of each 2-word entry contains a pointer to the window block of the task that has a file open. The window block contains pointers to areas on the file that are accessed by the task.

Whenever your task issues an I/O request, the system matches the appropriate physical device unit to the LUN that the call specifies. The system does this by indexing into the LUT by using the LUN number. Thus, if the call specifies 6 as the LUN, the system accesses the sixth 2-word entry in the LUT and associates the I/O request with the physical device unit to which the entry points. The number of LUN assignments valid for a task ranges from 0 to 255, but it cannot be greater than the number of LUNs specified at task-build time.

Figure 1–1 illustrates a typical LUT.

Figure 1–1: Logical Unit Table



ZK-4078-85

1.3.3 Changing LUN Assignments

Logical unit numbers have no significance until you associate a LUN with a physical device unit by using one of the following methods:

- At the time you build the task that is to do the I/O operation, you can specify an ASG (Assign) keyword option to the TKB. This option associates a physical device unit with a LUN referenced by the task being built.
- You or the system operator can issue a REASSIGN command to the Monitor Console Routine (MCR) or an ASSIGN/REDIRECT command to DCL. This command reassigns a LUN to another physical device unit and thus changes the correspondence between the LUN and the physical device unit. Note that this reassignment has no effect on the in-core image of a task.
- At run time, a task can dynamically change a LUN assignment by issuing the Assign LUN Executive directive macro (ALUN$). This changes the association of a LUN with a physical device unit during task execution.

1.4 Issuing an I/O Request

Your tasks perform I/O in the operating systems by submitting requests for I/O service as Queue I/O (QIO$) or Queue I/O and Wait (QIOW$) Executive directive macros. See the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual* for a complete description of system directives.

The RSX-11M-PLUS and Micro/RSX operating systems have a set of system macros that make issuing QIO$ macros easier. You must make these macros available to the source program by placing the MACRO-11 assembler directive .MCALL in the source program. The macros reside in the System Macro Library (LB:[1,1]RSXMAC.SML). Section 1.6.7 describes the function of .MCALL.

In both operating systems, as in most multiprogramming systems, tasks do not normally access physical device units directly. Instead, they use I/O services that the Executive provides, because the system can effectively multiplex the use of physical device units over many tasks. The Executive routes I/O requests to the appropriate device driver and queues them by the priority of the requesting task. I/O operations proceed concurrently with other activities in RSX-11M-PLUS and Micro/RSX systems.

Before the Executive queues a QIO$ request to the driver, the QIO$ must pass a series of tests executed by the Executive. If the request fails, the Executive rejects it. The Executive signals this rejection by setting the C-bit. As good programming practice, you should check for directive rejection by following the QIO$ macro with a MACRO-11 BCS instruction or its equivalent.

After the Executive queues an I/O request, the system does not wait for the operation to complete. Perhaps, the task that issued the QIO$ request cannot proceed until the I/O operation completes. In this case, the task should specify an event flag (see Section 1.4.1.4) in the QIO$ request and should issue a Wait-for Single Event Flag (WTSE$) Executive macro that specifies the same event flag at the point where synchronization must occur. Your task then waits for the I/O to complete by waiting for the Executive to set the specified event flag.

The QIO$ and Wait (QIOW$) macro is a more economical way to achieve this synchronization. QIOW$ waits until the system completes the I/O before returning control to the task. Thus, the additional WTSE$ macro is not necessary.

Each QIO$ or QIOW$ macro must supply sufficient information to identify and queue the I/O request. You may also want to include locations in your task to receive error or status codes and to specify the address of an asynchronous system trap (AST) service routine. Certain types of I/O operations require the specification of device-dependent information as well. Typical QIO$ parameters are the following:

- I/O function to be performed
- LUN associated with the physical device unit to be accessed
- Optional event flag number for synchronizing I/O completion processing (required for QIOW$)
- Optional address of the I/O status block (IOSB) to which the Executive returns information indicating successful or unsuccessful completion
- Optional address of an AST service routine in your task to be entered upon completion of the I/O request
- Optional device- and function-dependent parameters specifying such items as the starting address of a data buffer, the size of the buffer, and a block number

Several of the first six parameters in the QIO$ macro are optional, but you must reserve space for these parameters. During expansion of a QIO$ macro, the Executive defaults to a value of 0 for all null (omitted) parameters. Inclusion of the device- and function-dependent parameters depends on the physical device unit and function you specify. If you want to specify only an I/O function code, a LUN, and an address for an AST service routine, issue the following:

```
QIO$    IO.ATT,6,,,,ASTOX
```

IO.ATT is the QIO$ function code and the following describes the meaning of the parameters:

Parameter	Meaning
IO.ATT	Specifies the I/O function code for attach.
6	Specifies the LUN associated with the device unit.
,,,	Specifies null arguments for the event flag number, the request priority, and the address of the IOSB.
ASTOX	Specifies the AST address using the symbolic name ASTOX.

The system requires no additional device- or function-dependent parameters for an attach function. Section 1.6 describes the three legal forms of the macro.

For convenience, you may omit any commas if no parameters appear to the right of it. Therefore, if you did not want the AST, you could issue the preceding command as follows:

```
QIO$    IO.ATT,6
```

All extra commas have been dropped. However, if a parameter appears to the right of any place-holding comma, that comma must be retained.

1.4.1 QIO$ Macro Format

The arguments for a specific QIO$ macro call may be different for each I/O device your task accesses and for each I/O function it requests. However, the general format of the call is common to all devices.

Format

QIO$ fnc,lun,[efn],[pri],[isb],[ast],[<p1,p2,...,p6>]

1.4.1.1 Syntax Elements: Square Brackets, Angle Brackets, and Braces

Table 1–1 lists the syntax elements that you may use in macro call. Note that some of the syntax elements are required syntax for macro calls.

Table 1–1: Macro Syntax Elements

Element	Meaning
[ ]	Square brackets enclose optional parameters. You may use one or more of the optional parameters.
< >	Angle brackets must enclose function-dependent parameters if the QIO$ requires the parameters <p1,...,p6> . The angle brackets are part of the syntax and must be used. The parameters may or may not be present in a given QIO$ macro, and, if present, some may be optional.
{ }	Braces indicate that you must make a choice among the arguments enclosed within the braces.

The following paragraphs summarize the use of each QIO$ parameter. Section 1.7 explains different forms of the QIO$ macro itself.

1.4.1.2 FNC Parameter

The fnc parameter is the symbolic name of the I/O function that you want to request. This name is usually of the following form:

```
IO.xxx
```

The xxx parameter identifies the particular I/O operation.

For example, a QIO$ request to attach the physical device unit associated with a LUN specifies the function code IO.ATT with its complete QIO$ form appearing as follows:

```
QIO$  IO.ATT,lun
```

The lun parameter is the number assigned to the physical device unit.

A QIO$ request to cancel (or kill) all I/O requests for a LUN that you specified begins like the following:

```
QIO$  IO.KIL,...
```

The system internally stores the fnc parameter, which you specify in the QIO$ request, as a function code in the high-order byte and as modifier bits in the low-order byte of a single word.

The function code is in the range 0 to 31_{10} and is a binary value that the system supplies to match the symbolic name specified in the QIO$ request.

The system object module library defines the correspondence between global symbolic names and function codes. The Task Builder (TKB) searches the library. You can obtain local symbolic definitions by using the FILIO$ and SPCIO$ macros, which reside in the System Macro Library and which are summarized in Appendix A.

Several similar functions may have identical function codes, and you may distinguish them only by their modifier bits. For example, the DECtape read logical forward and read logical reverse functions have the same function code. Although the function codes are the same, the system stores the modifier bits for these two operations.

1.4.1.3 LUN Parameter

The lun parameter represents the logical unit number (LUN) of the associated physical device unit that the I/O request is to access. The association between the physical device unit and the LUN is specific to the task that issues the I/O request, and the LUN reference is usually device independent. You begin an attach request to the physical device unit associated with LUN 14 by using the following command:

```
QIO$   IO.ATT,14.,...
```

Because each task has its own Logical Unit Table (LUT) in which the correspondence between the LUN and the physical device unit is established, the legality of a LUN parameter is specific to the task that includes this parameter in a QIO$ request. In general, the LUN must be in the following range:

```
0  < LUN  < number of LUNs in table
```

The number of LUNs specified in the LUT of a particular task cannot exceed 255.

1.4.1.4 EFN Parameter

The efn parameter is a number representing the event flag to be associated with the I/O operation. It is an optional parameter for inclusion in the QIO$ request. The specified event flag is cleared when the I/O request is queued and is set when the I/O operation has completed. If the task issues the QIOW$ macro, the Executive suspends task execution until the I/O completes. If the task issues the QIO$ macro (with no WTSE$ macro), task execution proceeds in parallel with the I/O. When the task continues to execute, it may test the event flag whenever it chooses by using the Read All Event Flags (RDAF$) Executive directive macro (if group-global event flags are not being used), the Read Extended Flags (RDXF$) Executive directive (for all event flags, including group-global event flags), or the Read Single Event Flag (RSEF$) Executive directive.

If you specify an event flag number, it must be in the range 1 to 96. If you do not want to specify an event flag, you can omit efn or supply it with a value of 0. Event flags 1 to 32 are local (specific to the issuing task); event flags 33 to 64 are global (shared by all tasks in the system). Event flags 65 to 96 are group-global event flags (shared by all tasks in the same user group). Flags 25 to 32 and 57 to 64 are reserved for use by system software. Within these bounds, you can specify event flags as desired to synchronize I/O completion and task execution. Sections 1.4.2 and 1.4.3 provide a more detailed explanation of significant events and event flags.

Note

If an event flag is not specified, the Executive treats the directive as if it were a simple QIO$ request.

1.4.1.5 PRI Parameter

The optional pri parameter is supplied only to make RSX-11M-PLUS and Micro/RSX QIO$ requests compatible with RSX-11D. Thus, you should use a value of 0 (or a null) for this parameter.

1.4.1.6 ISB Parameter

The optional isb parameter identifies the address of the IOSB associated with the I/O request. This block is a 2-word array in which a code is returned that represents the final status of the I/O request on completion of the operation. This code is a binary value corresponding to a symbolic name of the form IS.xxx (for successful returns) or IE.xxx (for error returns). The binary error code is returned to the low-order byte of the first word of the status block. The error code can be tested symbolically, by name. For example, the symbolic status IE.BAD is returned if a bad parameter is encountered. The following illustrates the examination of the I/O status block, IOSB, to determine the success of the I/O:

```
QIO$C   IO.ATT,14.,2,,IOSB
BCS     DIRERR
WTSE$C  2
        .
        .
        .
CMPB    #IS.SUC,IOSB
BNE     ERROR
susan
```

The system object module library defines the correspondence between global symbolic names and I/O completion codes. TKB searches this library. The IOERR$ macro, which resides in the System Macro Library, obtains local symbolic definitions (summarized in Appendix B).

On completion of the I/O operation, the system returns certain device-dependent information to the high-order byte of the first word of isb. If a read or write operation is successful, the second word is also significant. For example, in the case of a read function on a terminal, the system returns in the second word of the isb the number of bytes that you typed preceding a carriage return. If a magnetic tape unit is the device and you specified a write function, this number represents the number of bytes actually written. The status block can be omitted from a QIO$ request if you do not intend to test for successful completion of the request.

1.4.1.7 AST Parameter

The optional ast parameter specifies the address of a service routine to be entered when an asynchronous system trap (AST) occurs. If you want to interrupt your task to execute special code on completion of an I/O request, you can specify an AST routine in the QIO$ request. When the specified I/O operation completes, control branches to this routine at the software priority of the requesting task. The system then executes the asynchronous code beginning at address ast, similar to the way the system executes an interrupt service routine. If you do not want to perform asynchronous processing, you can omit the ast parameter or specify a value of 0 in the QIO$ macro call.

Section 1.4.5 discusses the use of ASTs, and the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual* describes traps in detail.

1.4.1.8 P1,P2,...,P6 Parameters

The additional QIO$ parameters <p1,p2,...,p6> depend on the particular function and device specified in the I/O request. Typical parameters may include I/O buffer address, I/O buffer length, and so on. You can include between zero and six parameters depending on the particular I/O function. Subsequent chapters of this manual describe rules for including these parameters and legal values.

1.4.2 Significant Events

A **significant event** is a change in system status that causes the Executive to reevaluate the eligibility of all active tasks to run. (For some significant events, specifically those in which the current task becomes ineligible to run, only those tasks of lower priority are examined.) A significant event is usually caused (either directly or indirectly) by an Executive directive issued from within a task. This manual is concerned with the significant event caused by an I/O completion.

Significant events are normally set by Executive directives by completing a function that you specified. A task uses event flags to recognize the occurrence of specific events.

1.4.3 Event Flags

Event flags are a means by which tasks recognize specific events. (Tasks also use ASTs to recognize specific events.) In requesting a system operation (such as an I/O transfer), a task may associate an event flag with the completion of the operation. When the event occurs, the Executive sets the specified flag.

Tasks distinguish one event from another by using 96 event flags. Each event flag has a corresponding unique event flag number (efn). Numbers 1 to 32 form a group of flags that are unique to each task and are set or cleared as a result of that task's operation. Numbers 33 to 64 form a second group of flags that are common to all tasks, hence their name "common flags." Common flags may be set or cleared as a result of any task's operation. The last eight flags in each group, local flags (25-32) and common flags (57-64), are reserved for use by the system. Numbers 65 to 96 form the third group of flags, known as "group-global event flags." You can use these flags in any application where common event flags can be used; however, only tasks running under User Identification Codes (UICs) containing the group code specified when the group-global event flags were created can use them. Eight Executive directives provide the

support for creating, setting, clearing, reading, and testing event flags. See the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual* for a description of these directives.

The following textual example illustrates the use of a common event flag to synchronize task execution:

A task issues a QIO$ macro with an efn parameter specified. A WTSE$ macro follows the QIO$ and specifies the same event flag number as an argument. The Executive clears the event flag when the Executive queues the I/O request. Then, the Executive blocks the task when the Executive executes the WTSE$ directive. The task remains blocked until a significant event is declared at the completion of the I/O request and the significant event sets the event flag. The task resumes when the appropriate event flag is set, and execution resumes at the instruction following the WTSE$ macro. Using these macros and an event flag in this way ensures that the task does not manipulate the data until all the I/O has completed.

Specifying an event flag does not mean that a WTSE$ macro must be issued. Event flag testing can be performed at any time. The purpose of a WTSE$ macro is to block the task execution until an indicated event occurs. Hence, it is not necessary to issue a WTSE$ macro immediately following a QIO$ macro, but a task that depends on a specific I/O operation to complete must issue the WTSE$ macro before continuing.

A task can issue a Stop For Single Event Flag (STSE$) macro instead of a WTSE$ macro. When this is done, an event flag condition not satisfied results in the task's being stopped instead of being blocked until the event flag is set. A blocked task still competes for memory resources at its running priority. A stopped task competes for memory resources at priority 0.

1.4.4 System Traps

System traps can interrupt task execution and can cause a transfer of control to another memory location for special processing. The Executive handles system traps. The traps are relevant only to the task in which they occur. To use a system trap, a task must contain a trap service routine, which is automatically entered when the trap occurs.

There are two types of system traps: synchronous and asynchronous. You can use both to handle error or event conditions, but they differ in their relation to the task that is running when the traps are detected. The system traps differ as follows:

Synchronous (SSTs) — Signal error conditions within the executing task. If the same instruction sequence were repeated, the same synchronous trap would occur at the same place in the task. Synchronous traps are fully described in the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual.*

Asynchronous (ASTs) — Signal the completion of an external event such as an I/O operation. An AST usually occurs as the result of initiating or completing an external event rather than as a program condition.

Although not able to distinguish execution of an SST routine from task execution, the Executive is aware that a task is executing an AST routine. An AST routine can be interrupted by an SST routine but not by another AST routine.

1.4.5 Asynchronous System Traps

The primary purpose of an AST is to inform the task that a certain event has occurred—for example, the completion of an I/O operation. As soon as the task has serviced the event, it can return to the interrupted code.

Some directives can specify both an event flag and an AST; with these directives, you can use ASTs as an alternative to event flags or you can use the two together. Therefore, you can specify the same AST routine for several directives, each with a different event flag. Thus, when the Executive passes control to the AST routine, the event flag can determine the action required. However, it is standard programming practice to use the I/O status block (IOSB) rather than the event flags to determine which I/O operation is completed. Thus, when control is passed to an AST from a QIO$, the IOSB is on top of the stack. Use this IOSB to determine which I/O has completed.

The Executive queues ASTs in a first-in/first-out (FIFO) queue for each task and monitors all asynchronous service routine operations. Because asynchronous traps may be the end result of I/O-related activity, the task cannot control the occurrence of the ASTs directly. An example of an asynchronous trap condition is the completion of an I/O request. The timing of such an operation clearly cannot be predicted by the requesting task. If the task does not specify an AST service routine in an I/O request, a trap does not occur and normal task execution continues.

However, the task may, under certain circumstances, block recognition of ASTs to prevent simultaneous access to a critical data region. When access to the critical data region has been completed, the queued ASTs may again be honored. The Disable AST Recognition (DSAR$S) and Enable AST Recognition (ENAR$S) Executive directives provide the mechanism with proper access to a critical data region.

Associating ASTs with I/O requests enables the requesting task to be truly event driven. The system executes the AST service routine contained in the initiating task as soon as possible, consistent with the task's priority. Using the AST routine to service I/O-related events provides a response time that is considerably better than a polling mechanism, and it provides for better overlap processing than the simple QIO$ and WTSE$ macros. ASTs also provide an ideal mechanism for use in multiple buffering of I/O operations.

The Executive inserts all ASTs in a FIFO queue on a per task basis as they occur (that is, the event that they are to signal has expired). The Executive executes them one at a time whenever the task does not have ASTs disabled and is not already in the process of executing an AST service routine. Executing the AST includes storing certain information on the task's stack, including the task's WTSE$ mask word and address, the Directive Status Word (DSW), the program status (SP), the program counter (PC), and any trap-dependent parameters. The task's general-purpose registers R0–R5 are not saved, and thus AST service routines must save and restore all registers used. If the registers are not restored after an AST has occurred, the task's subsequent execution may be unpredictable.

After an AST is processed, the trap-dependent parameters (if any) must be removed from the task's stack and an AST Service Exit ASTX$S macro must be executed. The ASTX$S macro, described in Section 1.6.10, issues the AST Service Exit directive. On AST service exit, control returns to another queued AST, to the executing task, or to another task waiting to run.

The *RSX-11M-PLUS and Micro/RSX Executive Reference Manual* describes in detail the purpose of AST service routines and all Executive directives that handle them.

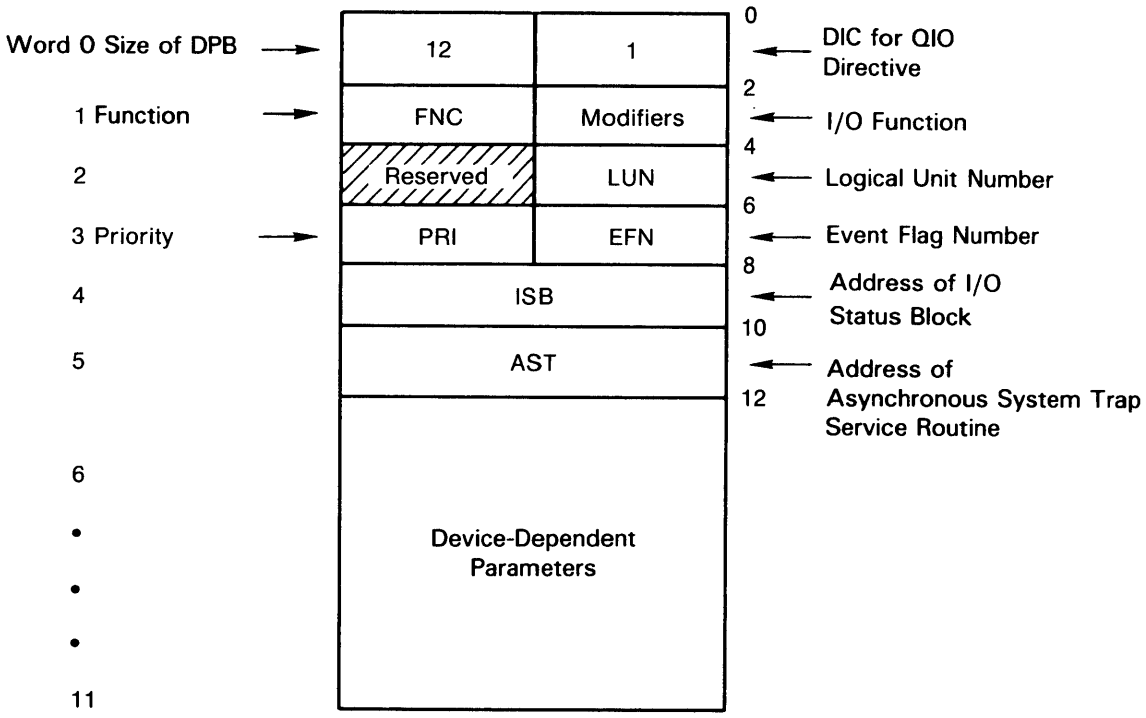
1.5 Directive Parameter Blocks

A Directive Parameter Block (DPB) is a fixed-length area of contiguous memory that contains the arguments that you specify in an Executive directive macro call. The DPB for a QIO$ directive has a length of 12 words. The Executive generates it as the result of expanding a QIO$ macro call. The first two bytes of the DPB contain the following:

- The first byte of the DPB contains the Directive Identification Code (DIC)—always 1 for QIO$.
- The second byte contains the size of the DPB in words—always 12 for RSX-11M-PLUS and Micro/RSX.

During the assembly of your task containing QIO$ requests, the MACRO-11 assembler generates a DPB for each I/O request specified in a QIO$ macro call. At run time, the Executive uses the arguments stored in each DPB to create, for each request, an I/O packet in system dynamic storage. Figure 1-2 illustrates the layout of a sample DPB.

Figure 1-2: QIO$ Directive Parameter Block



ZK-005-81

1.5.1 I/O Packets

The Executive enters the I/O packet by priority into a queue of I/O requests for the specified physical device unit. The Executive creates and maintains this queue and orders it by the priority of the tasks that issued the requests. The I/O drivers examine their respective I/O packet queues for the I/O request with the highest priority capable of being executed. The driver removes this packet from the queue and performs the I/O operation. The process is then repeated until the queue is empty of all requests.

1.5.2 Significant Event Declaration

After the I/O request has been completed, the Executive declares a significant event and may do one or more of the following:

- Set an event flag.
- Cause a branch to an AST service routine.
- Return the I/O status.

Any of the preceding actions depend on the arguments specified in the original QIO$ macro call.

1.6 I/O Related Macros

Both operating systems supply several system macros to issue and return information about I/O requests. These macros reside in the System Macro Library and must be made available during assembly by including the MACRO–11 assembler directive .MCALL in the task's code.

The RSX–11M–PLUS and Micro/RSX system also supplies FORTRAN-callable subroutines that perform the same functions as the system macros. See the *RSX–11M–PLUS and Micro/RSX Executive Reference Manual* for more details.

Most of the Executive directive macros described in this section have three distinct forms. The following sections summarize the forms of QIO$, but the characteristics of each form also apply to QIOW$, ALUN$, GLUN$, and the other described Executive directive macros.

1.6.1 QIO$ Form

The QIO$ form is useful for a directive operation that is to be issued several times from different locations in a non-reentrant program segment. The QIO$ form is most useful when the directive is issued several times with varying parameters (one or more but not all parameters change) or in a reentrant program section when a directive is issued several times even though the DPB is not modified. This form produces only the directive's DPB and must be issued from a data section of the program. The code for actually executing a directive in the QIO$ form is produced by a special macro, DIR$.

Because execution of the directive is separate from the creation of the directive's DPB, the following occur:

- A QIO$ form of a given directive needs to be issued only once (to produce its DPB).
- A DIR$ macro associated with a given directive can be issued several times without incurring the cost of generating a DPB each time it is issued.

- It is easy to access and change the directive's parameters by labeling the start of the DPB and by using the offsets defined by the directive.

When a program issues the QIO$ form of a macro call, the parameters required for DPB construction must be valid expressions for MACRO–11 data storage instructions (such as .BYTE, .WORD, and .RAD50). You can alter individual parameters in the DPB. You might do this if you want to use the directive many times with varying parameters.

1.6.2 QIO$S Form

Program segments that need to be reentrant should use the QIO$S form. Only the QIO$S form produces the DPB at run time. The other two forms produce the DPB at assembly time.

In this form, the macro produces code to push a DPB onto the stack. The code is followed by an EMT 377 instruction. In this case, the parameters must be valid source operands for MOV-type instructions. For a 2-word Radix–50 name parameter, the argument must be the address of a 2-word block of memory containing the name. Note that you should not use the stack pointer (or any reference to the stack pointer) to address directive parameters when the QIO$S form is used.[1]

Note that in the QIO$S form of the macro, the macro arguments are processed from right to left. Therefore, when using code in the following form, the result may be obscure.

```
MACRO$S,,(R4)+,(R4)+
```

1.6.3 QIO$C Form

Use the QIO$C form when a directive is to be issued only once. The QIO$C form eliminates the need to push the DPB (created at assembly time) onto the stack at run time. Other parts of the program, however, cannot access the DPB because the DPB address is unknown.

The QIO$C form generates a DPB in a separate program section[2] called $DPB$$. The DPB is first followed by a return to the user-specified program section, then by an instruction to push the DPB address onto the stack, and finally by an EMT 377 instruction. To ensure that the program reenters the correct program section, you must specify the program section name in the argument list immediately following the DPB parameters. If the argument is not specified, the program reenters the blank (unnamed) program section.

This form also accepts an optional final argument that specifies the address of a routine to be called (by a JSR instruction) if an error occurs during the execution of the directive (See the *RSX–11M–PLUS and Micro/RSX Executive Reference Manual* for more information).

When a program issues the QIO$C form of a macro call, the parameters required for DPB construction must be valid expressions for MACRO–11 data storage instructions (such as .BYTE, .WORD, and .RAD50). (This is not true for the program-section argument and the error-routine argument, which are not a part of the DPB.)

---

[1] Subroutine or macro calls can use the stack for temporary storage, thereby destroying the positional relationship between SP and the parameters.

[2] Refer to the *PDP–11 MACRO–11 Language Reference Manual* for a description of program sections.

1.6.3.1 Additional QIO Macro Call Information

Parameters for both the QIO$ and QIO$C forms of the macro must be valid expressions for the MACRO–11 .WORD and .BYTE statements. Parameters for the QIO$S form must be valid source operand address expressions for MACRO–11 assembler instructions such as MOV and MOVB. The same parameters are noted in the three distinct forms of the macro call that follow.

Format

```
QIO$   IO.RLB,6,2,,,AST01, <RDBUF,80.>
QIO$C  IO.RLB,6,2,,,AST01, <RDBUF,80.>
QIO$S  #IO.RLB,#6,#2,,,#AST01, <#RDBUF,#80.>
```

Only the QIO$S form of the macro produces the DPB dynamically. The other two forms generate the DPB at assembly time. The *RSX–11M–PLUS and Micro/RSX Executive Reference Manual* describes the characteristics and use of these different forms.

The sections that follow describe the following Executive directives and MACRO–11 assembler macros:

QIO$	Requests an I/O operation and supplies parameters for that request.
QIOW$	Equivalent to QIO$ followed by WTSE$.
DIR$	Specifies the address of a DPB as its argument and generates code to execute the directive.
.MCALL	Makes all macros referenced during task assembly available from the System Macro Library.
ALUN$	Associates a LUN with a physical device unit at run time.
GLUN$	Requests that the information about a physical device unit to LUN association be returned to a buffer that you specify.
ASTX$S	Terminates execution of an AST service routine.
WTSE$	Instructs the system to block execution of the issuing task until a specified event flag is set.

1.6.4 The QIO$ Macro: Issuing an I/O Request

As previously described, you may use three general forms of the QIO$ macro. They are reviewed as follows:

- QIO$ generates only the DPB for the I/O request. This form of the macro call is used with DIR$ (see Section 1.6.6) to execute an I/O request.
- QIO$S generates a DPB for the I/O request on the stack as well as generating code to execute the request.
- QIO$C generates a DPB and code, but the DPB is generated in a separate program section.

1.6.5 The QIOW$ Macro: Issuing an I/O Request and Waiting for an Event Flag

The QIOW$ macro is equivalent to a QIO$ macro followed by a WTSE$ macro. It is more economical to issue a QIOW$ request than to use the two separate macros. An event flag (efn parameter) must be specified with QIOW$.

Note

Please note that tasks or applications that execute many I/O operations will run much more efficiently using QIOW$ rather than QIO$ followed by a WTSE$. Efficiency increases because system overhead is reduced.

Format

QIOW$ function,lun,efn,[pri],[isb],[ast],[<p1,...,p6>]

See the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual* for a complete description of the QIOW$ macro.

1.6.6 The DIR$ Macro: Executing a Directive

The DIR$ (execute directive) macro allows a task to reference a previously defined Directive Parameter Block (DPB).

Format

DIR$ [addr],[err]

Parameters

addr

Specifies the address of a DPB used in the directive. If addr is not included, the DPB itself or the address of the DPB is assumed to already be on the stack. This parameter must be a valid source operand for a MOV instruction generated by the DIR$ macro.

err

Indicates an optional argument that specifies the address of an error routine to which control branches if the directive is rejected. The branch occurs by means of a Jump to Subroutine Program Counter (JSR PC) instruction (err, if the C-bit is set), indicating rejection of the QIO$ directive.

Example

```
QIOREF: QIO$    IO.RLB,6,2,,,AST01,<BUFFER,80.>  ;CREATE QIO$ DPB
        .
        .
        .
READ1:  DIR$    #QIOREF                  ; ISSUE I/O REQUEST
        .
        .
        .
READ2:  DIR$    #QIOREF                  ; ISSUE I/O REQUEST
```

Shows that the DIR$ macro actually generates the code to execute the QIO$ directive. It

provides no QIO$ parameters of its own, but it references the QIO$ DPB at address QIOREF by supplying this label as an argument.

1.6.7 The .MCALL Directive: Retrieving System Macros

The .MCALL directive is a MACRO–11 assembler directive that retrieves macros from the System Macro Library (LB:[1,1]RSXMAC.SML) for use during assembly. You must include it in every task that invokes system macros. The .MCALL directive is usually placed at the beginning of your task source module and specifies, as arguments in the call, all system macros that must be made available to your task from the library.

Example

```
        .MCALL  QIO$,QIO$S,DIR$,WTSE$S          ; MAKE MACROS AVAILABLE
                .
                .
                .
ATTACH: QIO$S   #IO.ATT,#6,,,#IOSB,#ASTO2        ; ATTACH DEVICE
                .
                .
                .
QIOREF: QIO$    IO.RLB,6,,,IOSB,ASTO1,...        ; CREATE ONLY QIO$ DPB
                .
                .
                .
READ1:  DIR$    #QIOREF,DIRERR                   ; ISSUE I/O REQUEST
                .
                .
                .
```

Illustrates the use of the .MCALL directive. You can include as many macro references as can fit on a line in a single .MCALL directive. You can specify any number of .MCALL directives.

1.6.8 The ALUN$ Macro: Assigning a LUN

The Assign LUN (ALUN$) macro associates a logical unit number (LUN) with a physical device unit at run time. All three forms of the macro call may be used. Assign LUN does not request I/O for the physical device unit, nor does it attach the unit for exclusive use by the issuing task. It only establishes a LUN–physical device unit relationship, so that when the task requests I/O for that particular LUN, the task can reference the associated physical device unit. You can issue the macro from a MACRO–11 program by using the format shown next.

Format

ALUN$ lun,dev,unt

Parameters

lun

Specifies the logical unit number to be associated with the specified physical device unit. See Sections 1.3 and 1.4.1.3.

dev

Specifies the device name of the physical device or a logical device name assigned to a physical device (see the MCR command ASN or the DCL command ASSIGN).

Note

When specifying a device name for the dev parameter you must use uppercase letters.

unt

Specifies the unit number of the preceding device specified.

For example, to associate LUN 10. with terminal unit 2, a task could issue the following macro call:

```
ALUN$C  10.,TT,2
```

A unit number of 0 represents unit 0 for multiunit devices such as disk, DECtape, or terminals; it indicates the single available unit for devices without multiple units, such as card readers and line printers.

Logical devices are included as part of RSX–11M–PLUS and Micro/RSX.

Example

```
;
; DATA DEFINITIONS
;
ASSIGN: ALUN$    10.,TT,2       ; GENERATE DPB
        .
        .
        .
;
; EXECUTABLE SECTION
;
        DIR$     #ASSIGN        ; EXECUTE DIRECTIVE
        .
        .
        .
        ALUN$C   10.,TT,2       ; GENERATE DPB IN SEPARATE PROGRAM
        .                       ; SECTION, THEN GENERATE CODE TO
        .                       ; EXECUTE THE DIRECTIVE
        .
        ALUN$S   #10.,#"TT,#2   ; GENERATE DPB ON STACK, THEN
                                ; EXECUTE DIRECTIVE
```

Illustrates the use of the three forms of the ALUN$ macro.

1.6.8.1 Physical Device Names

Table 1–2 contains physical device names, listed alphabetically, that you may include as dev parameters. Note, that, not all the devices are supported by Micro/RSX. Devices supported by Micro/RSX are identified in a separate column.

Table 1–2: Physical Device Names

Device Mnemonic	Device	Supported by Micro/RSX
BS	DT03/DT07 UNIBUS switch	No
CD	CD11 card reader	No
CP	Central processing unit (CPU) in a multiprocessor system	No
CR	CR11/CM11 card reader	No
CT	TA11/TU60 tape cassette	No
DB	RP04, RP05, RP06 disk pack	No
DD	TU58 DECtape II	Yes
DF	RF11/RS11 fixed-head disk	No
DK	RK11/RK05 cartridge disk	No
DL	RL11/RLV11/RL01/RL02 cartridge disk	Yes
DM	RK611/RK06 and RK711/RK07 cartridge disk	No
DP	RP11/RP02/RP03 disk pack	No
DR	RM02/RM03/RM05 disk pack and RM80/RP07 fixed-media disk	No
DS	RS03 and RS04 fixed-head disks	No
DT	TC11/TU56 DECtape	No
DU	RA80/RA81/RA82 fixed-media disk, RA60 disk pack, RC25 disk subsystem, RD51/RD52/RD53/RD54 fixed-media disk, RUX50 UNIBUS interface, RQDX50 Q-bus interface, and RX50/RX33 flexible disk	Yes
DX	RX11/RX01 flexible disk	No
DY	RX211/RX02 flexible disk	Yes
EM	ML-11 fast electronic mass-storage device	No
LA	LPA11-K laboratory peripheral accelerator	No
LP	LP11/LS11/LV11 and the KMC-11-A auxiliary processor	No
LP	LA180/LN01/LN03/LP25/LP26 and the LPV11 controller	Yes
LR	PCL11-A/PCL11-B receiver port	No

Table 1–2 (Cont.): Physical Device Names

Device Mnemonic	Device	Supported by Micro/RSX
LT	PCL11-A/PCL11-B transmitter port	No
MM	TU16/TE16/TU45/TU77/TM02/TM03 magnetic tape	No
MS	TS11 and the TU80 magnetic tape device	No
MS	TSV05 and the TK25 magnetic tape	Yes
MT	TM11/TU10/TU11 or TS03 magnetic tape	No
MU	TK50 cartridge tape device	Yes
MU	TU81 magnetic tape device	Yes
NL	The null device	Yes
PP	PC11 paper tape punch	No
PR	PC11 or PR11 paper tape reader	No
TT	Terminals (regardless of interface) (not Network Command Terminals)	Yes
XE	QIO DEUNA driver	No
XM	DMC11 synchronous communication line interface	No
XW	DUP11 synchronous communication line interface	No
JA-JZ	Reserved for customer use (not used by DIGITAL)	
QA-QZ	Reserved for customer use (not used by DIGITAL)	
ZA-ZZ	Reserved for customer use (not used by DIGITAL)	

1.6.8.2 Pseudo-Device and Physical Device Names

A pseudo-device name is a logical device name that must be directed to a physical device unit. A pseudo-device name can be redirected, by the operator, to another physical device at any time without requiring changes in programs that reference the pseudo-device name. (The DV.PSE bit in the LUN information buffer is set to 1 if a pseudo-device name references a physical device.) Dynamic redirection of a physical device unit affects all tasks; MCR command REDIRECT affects only one task.

Nonphysical device names are not associated with a physical device but with a driver that interfaces with data structures instead of a real physical device.

The following list indicates the pseudo devices (and nonphysical devices) supported by RSX–11M–PLUS and Micro/RSX:

Nonphysical Name	Physical Name	Driver	Unit
	CL (pseudo)		Console listing, normally the line printer.
	CO (pseudo)	CODRV	Console output, normally the main operator's console.
HT		HTDRV	Network remote terminal.
	LB (pseudo)		System library device, normally the device from which the system was bootstrapped. For example, tasks such as TKB and MAC access the LB device for default library files.
NL		NLDRV	Null device.
NS			Network pseudo device for NSP.
NX			Network pseudo device for DLX.
RD		RDDRV	Online reconfiguration pseudo device.
RT		RTDRV	Network Command Terminals (NCTs).
	SP (pseudo)		Spooling scratch disk device.
	SY (pseudo)		Your system default device. On nonmultiuser systems, SY is normally the disk from which the system was bootstrapped. On multiuser systems, SY is normally the default login device.
	TI (pseudo)		Pseudo input terminal; TI0 is the terminal from which a task was requested. The pseudo device TI cannot be redirected, because such redirection would have to be handled on a per-task rather than a systemwide basis (that is, you can change the TI device for one task without affecting the TI assignments for other tasks).
VT		VTDRV	Virtual terminal. Used by some RSX-11M-PLUS and Micro/RSX offspring tasks as TI for command and data I/O.

1.6.9 The GLUN$ Macro: Retrieving LUN Information

The Get LUN Information (GLUN$) macro requests the return of information about association between a LUN and physical device unit in a 6-word buffer specified by the issuing task. Upon successful completion of a QIO$ directive, the buffer contains the information listed in Table 1–3, as appropriate for the specific device. All three forms of the macro call may be used. It is issued from a MACRO–11 program by using the format shown next.

Format

```
GLUN$  lun,buf
```

Parameters

lun

Specifies the logical unit number associated with the physical device unit for which information is requested. See Sections 1.3 and 1.4.1.3.

buf

Specifies the 6-word buffer to which information is returned.

For example, to request information on the disk unit associated with LUN 8, issue the following call:

```
GLUN$C   8.,IOBUF
```

Table 1–3: Get LUN Information

Numerical Offset			*-3 Symbolic Offset			
Word	**Byte**	**Bit**	**Word**	**Byte**	**Bit**	**Contents**
0			G.LUNA			Name of device associated with LUN (ASCII bytes)
1	0			G.LUNU		Unit number of associated device
	1			G.LUFB		Driver flag value. Returned as 128_{10} or 200_8 if the driver is resident, or as 0 if a loadable driver is not in the system
2			G.LUCW[1]			First device characteristics word:
		0	(U.CW1)		(DV.REC)	Unit record-oriented device (for example, card reader, line printer) (1 = yes)
		1			(DV.CCL)	Carriage-control device (for example, line printer, terminal) (1 = yes)
		2			(DV.TTY)	Terminal device (1 = yes)
		3			(DV.DIR)	Directory device (for example, DECtape, disk) (1 = yes)
		4			(DV.SDI)	Single directory device (for example, ANSI-standard magnetic tape) (1 = yes)
		5			(DV.SQD)	Sequential device (for example, ANSI-standard magnetic tape) (1 = yes)
		6			(DV.MSD)	Mass-storage device (for example, disks and tapes) (1 = yes)
		7			(DV.UMD)	User-mode diagnostics supported (1 = yes)

[1] The following word and bit symbols shown in parentheses are used in defining and referencing corresponding items in the device Unit Control Block (UCB).

Table 1–3 (Cont.): Get LUN Information

Numerical Offset			*-3 Symbolic Offset			
Word	**Byte**	**Bit**	**Word**	**Byte**	**Bit**	**Contents**
		8			(DV.EXT)	Device supports 22-bit direct addressing
		9			(DV.SWL)	Unit software write-locked (1 = yes)
		10			(DV.ISP)	Input spooled device (1 = yes)
		11			(DV.OSP)	Output spooled device (1 = yes)
		12			(DV.PSE)	Pseudo device (1 = yes)
		13			(DV.COM)	Device mountable as a communications channel for Digital network support (for example, DP11, DU11) (1 = yes)
		14			(DV.F11)	Device mountable as a Files–11 device (for example, disk or DECtape) (1 = yes)
		15			(DV.MNT)	Device mountable (logical OR of bits 13 and 14) (1 = yes)
3			G.LUCW+02			Second device characteristics word:
			(U.CW2)	(U2.xxx)		Device-specific information
4			G.LUCW+04			Third device characteristics word:
			(U.CW3)	(U3.xxx)		Device-specific information [2]
5			G.LUCW+06			Fourth device characteristics word:
			(U.CW4)			Default buffer size (for example, for disks, and line length for terminals)

[2]For mass-storage devices, such as disks, DECtape, and DECtape II, this is the number of blocks (maximum logical block number plus one). For the proper use of the RX211/RX02 flexible disk, you must test G.LUCW+04 to determine the media density.

Example

```
;
; DATA DEFINITIONS
;
GETLUN: GLUN$    6,DSKBUF        ; GENERATE DPB
          .
          .
          .

;
; EXECUTABLE SECTION
;
```

```
DIR$     #GETLUN          ; EXECUTE DIRECTIVE
.
.
.
GLUN$C   6,DSKBUF         ; GENERATE DPB IN SEPARATE PROGRAM
.                         ; SECTION, THEN GENERATE CODE TO
.                         ; EXECUTE THE DIRECTIVE
.
GLUN$S   #6,#DSKBUF       ; GENERATE DPB ON STACK, THEN
                          ; EXECUTE DIRECTIVE
```

Illustrates the use of the three forms of the GLUN$ macro.

1.6.10 The ASTX$S Macro: Terminating AST Service

The ASTX$S macro terminates execution of an asynchronous system trap (AST) service routine. The Executive provides all forms of the macro. However, the S-form requires less space and executes at least as fast as the ASTX$ or ASTX$C form of the macro.

Format

ASTX$S [err]

Parameter

err

Indicates an optional argument that specifies the address of an error routine to which control branches if the directive is rejected.

After the Executive completes the operation specified in this macro call, the Executive executes the next AST immediately if another AST is queued and ASTs have not been disabled. Otherwise, the Executive restores the task's state existing before the AST was entered. (The AST service routine must save and restore the registers it uses.)

1.6.11 The WTSE$ Macro: Wait-for Single Event Flag

The WTSE$ macro suspends execution of the issuing task until the Executive sets the event flag specified in the macro call. This macro is extremely useful in synchronizing other activity with the completion of I/O operations. You may use all three forms of the WTSE$ macro call.

Format

WTSE$ efn

Parameter

efn

Specifies the event flag number.

WTSE$ blocks the task from execution until the specified event flag is set. Frequently, you may include an efn parameter in a QIO$ macro call, and the Executive sets the event flag upon the completion of the I/O operation specified in that call.

Example

```
;
        .MCALL  WTSE$, ALUN$S, QIO$C, DIR$
        .MCALL  QIO$S, WTSE$S, WTSE$C
; DATA DEFINITIONS
;
WAIT:   WTSE$  5              ; GENERATE DPB
IOSB:   .BLKW  2              ; I/O STATUS BLOCK
        .
        .
        .
;
; EXECUTABLE SECTION
;
        ALUN$S #14.,#"MM      ; ASSIGN LUN 14 TO MAGNETIC
                              ; TAPE UNIT ZERO
        QIO$C   IO.ATT,14.,5  ; ATTACH DEVICE
        DIR$    #WAIT         ; EXECUTE WAIT FOR DIRECTIVE
        .
        .
        .
        QIO$S   #IO.RLB,#14.,#2,,#IOSB,,<#BUF,#80.>
        .                     ; READ RECORD, USE EFN2
        .
        .
        WTSE$S  #2            ; WAIT FOR READ TO COMPLETE
        .
        .
        .
        QIO$C   IO.WLB,14.,3,,IOSB,,<BUF,80.>
        .                     ; WRITE RECORD, USE EFN3
        .
        .
        WTSE$C  3             ; WAIT FOR WRITE TO COMPLETE
        .
        .
        .
        QIO$C   IO.DET,14.    ; DETACH DEVICE
        .
        .
        .
```

Illustrates task blocking until the specified event flag is set. This example also shows the use of three forms of the WTSE$ macro call.

1.7 Standard I/O Functions

You can specify a large number of I/O operations with the QIO$ macro. You can request a particular operation by including the appropriate function code as the first parameter of a QIO$ macro call. Certain functions are standard. These functions are almost totally device independent; thus, you can request them for nearly every device described in this manual. Other I/O functions are device dependent and are specific to the operation of only one or two I/O devices. This section summarizes the function codes and characteristics of the following standard device-independent I/O operations:

- Attaching to an I/O device
- Detaching from an I/O device
- Canceling I/O requests
- Reading a logical block
- Reading a virtual block
- Writing a logical block
- Writing a virtual block

For certain physical device units, a standard I/O function may be described as being a no operation (no-op). This means that no operation occurs as a result of specifying the function, and the Executive returns an I/O status code of IS.SUC in the I/O status block (IOSB) specified in the QIO$ macro call.

1.7.1 I/O Subfunction Bits

Most terminal QIO$ functions can be modified by using the symbolic name of a subfunction bit in a logical OR with the QIO$ function. The symbolic names of subfunction bits take the form TF.xxx, where **xxx** is the acronym of the subfunction to be performed. A standard QIO$ function called IO.ATT (attach a device) in a logical OR with the TF.ESQ subfunction for terminals (recognize escape sequences) would appear as follows:

```
QIO$C IO.ATT!TF.ESQ,lun,[efn],[pri],[isb],[ast]
```

A subfunction bit modifies and extends the operation indicated by the terminal QIO$ function. Note that the use of TF.ESQ with IO.ATT is a terminal-specific function. Often, you may want to use more than one subfunction bit when you use QIO$ requests to read or write to a terminal. In this case, you may use several subfunction bits together in a logical OR. The standard QIO$ IO.ATT function may be extended to both recognize escape sequences and allow special processing in the task upon the occurrence of ASTs. To do this requires that you combine in a logical OR two subfunction bits with the IO.ATT function. If you do this, the QIO$ IO.ATT macro would appear as follows:

```
QIO$C IO.ATT!TF.ESQ!TF.AST,lun,[efn],[pri],[isb],[ast]
```

Note that the use of TF.ESQ or TF.AST with IO.ATT is a terminal-specific function.

If your task invokes a subfunction bit that is not supported on the system, the subfunction bit may be ignored or an error may be issued by the system and the QIO$ rejected.

The subfunction bits that apply to a specific QIO$ macro are described with that QIO$ macro in Chapter 2.

1.7.2 QIO$C IO.ATT—Attaching to an I/O Device

Use the IO.ATT function code when your task requires exclusive use of an I/O device.

Successful completion of an IO.ATT request exclusively dedicates the specified physical device unit to the task that issues the IO.ATT. This enables the task to process input or output in an unbroken stream and is especially useful on sequential, non-file-oriented devices such as terminals, card readers, and line printers. An attached physical device unit remains under control of the task until that task explicitly detaches it. To detach the device, the task issues the QIO$C IO.DET macro with the logical unit number (LUN) previously assigned to the attached device.

While a task attaches a physical device unit, the I/O driver for that unit dequeues only I/O requests issued by the task that attaches the unit. However, a privileged task can issue a write breakthrough function (IO.WBT) to a terminal attached by another task. This is an exception for terminals only. Thus, except for the case of IO.WBT, the Executive does not process a request to attach a device unit already attached by another task, or until the attachment by the first task is broken and no higher-priority request exists for the attached unit.

A LUN that is associated with an attached physical device unit may not be reassigned by an Assign LUN (ALUN$) macro unless at least one LUN is still assigned to the attached device. If the task that issued an attach function exits or is aborted before it issues a corresponding detach function, the Executive detaches the physical device unit.

Format

```
QIO$C   IO.ATT,lun,[efn],[pri],[isb],[ast]
```

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.

pri

Makes RSX-11M-PLUS and Micro/RSX QIO$ requests compatible with RSX-11D. Thus, a value of 0 (or a null) should be used for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Section 1.4.1.6.

ast

Specifies the address of a service routine to be entered for IO.ATT when the IO.ATT operation completes. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting

task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.

See the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual* for further details on ASTs.

1.7.3 QIO$C IO.DET—Detaching from an I/O Device

IO.DET detaches a physical device unit that has been previously attached by an IO.ATT request. You can issue the QIO$C IO.DET macro in the format shown next.

Format

QIO$C IO.DET,lun,[efn],[pri],[isb],[ast]

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.

pri

Makes RSX-11M-PLUS and Micro/RSX QIO$ requests compatible with RSX-11D. Thus, a value of 0 (or a null) should be used for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Section 1.4.1.6.

ast

Specifies the address of a service routine to be entered when an asynchronous system trap (AST) occurs. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.

Example

```
.MCALL  ALUN$S,QIO$S
ALUN$S  #14.,#"LP          ; ASSOCIATE LINE PRINTER WITH LUN 14
.
.
.
QIO$S   #IO.ATT,#14.       ; ATTACH LINE PRINTER
.
.
.
```

```
LOOP:   QIO$S   #IO.WLB,#14.,...  ;PRINT
        .
        .
        .
        QIO$S   #IO.DET,#14.      ; DETACH LINE PRINTER
```

Illustrates that the LUN specifications of both IO.ATT and IO.DET must be the same. This example also illustrates the use of S-forms of several macro calls.

1.7.4 QIO$C IO.KIL—Canceling I/O Requests

IO.KIL cancels the issuing task's I/O requests for a particular physical device unit.

For I/O requests waiting for service (that is, in the I/O driver's queue), the Executive returns a status code of IE.ABO in the I/O status block (IOSB). An event flag is set, if specified. But any AST service routine that you may have specified is not executed.

For I/O requests being processed by any I/O driver, except the disk or DECtape drivers, the Executive returns the IE.ABO status code. The Executive also returns other status information (byte count, and so on) in the I/O status block. An AST, if specified, is executed.

If your task issues an IO.KIL for disk, DECtape, or DECtape II I/O requests being processed, the IO.KIL acts as a no-op. The I/O request completes, except in the case in which a DECtape transfer is blocked by a select error. Because disk and DECtape operate quickly, IO.KIL causes the return of IS.SUC in the IOSB.

IO.KIL is useful in such special cases as canceling an I/O request on a physical device unit from which a response is overdue (for example, a read on a paper tape reader).

Format

QIO$C IO.KIL,lun,[efn],[pri],[isb],[ast]

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.

pri

Makes RSX-11M-PLUS and Micro/RSX QIO$ requests compatible with RSX-11D. Thus, a value of 0 (or a null) should be used for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Section 1.4.1.6.

ast

Specifies the address of a service routine to be entered when an asynchronous system trap (AST) occurs. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.

1.7.5 QIO$C IO.RLB—Reading a Logical Block

Issue IO.RLB to read a block of data from the specified physical device unit.

Format

QIO$C IO.RLB,lun,[efn],[pri],[isb],[ast], <stadd,size,pn>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.

pri

Makes RSX-11M-PLUS and Micro/RSX QIO$ requests compatible with RSX-11D. Thus, a value of 0 (or a null) should be used for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Section 1.4.1.6.

ast

Specifies the address of a service routine to be entered when an asynchronous system trap (AST) occurs. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.

stadd

Specifies the starting address of the data buffer. The address must be word aligned for certain drivers; otherwise, stadd may be on a byte boundary.

size

Specifies the size of the stadd buffer in bytes. The buffer must be within the task's address space.

pn

Specifies one to four optional parameters that specify such additional information as block numbers for certain devices.

1.7.6 QIO$C IO.RVB—Reading a Virtual Block

IO.RVB reads a virtual block of data from the specified physical device unit. A "virtual" block indicates a relative block position within a file and is identical to a logical block for such sequential, record-oriented devices as terminals and card readers. For these sequential, record-oriented devices, the Executive converts IO.RVB to IO.RLB before it issues the QIO$.

Note

Any subfunction bits specified in the IO.RVB request are stripped off in this conversion.

All tasks should use virtual rather than logical reads to file-structured devices. However, if a task issues a virtual read for a file-structured device (disk, DECtape, or DECtape II), you must ensure that a file is open on the specified physical device unit.

Format

QIO$C IO.RVB,lun,[efn],[pri],[isb],[ast], <stadd,size,pn>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.

pri

Makes RSX-11M-PLUS and Micro/RSX QIO$ requests compatible with RSX-11D. Thus, a value of 0 (or a null) should be used for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Section 1.4.1.6.

ast

Specifies the address of a service routine to be entered when an asynchronous system trap occurs. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.

stadd

Specifies the starting address of the data buffer. The address must be word aligned for certain drivers; otherwise, stadd may be on a byte boundary.

size

Specifies the size of the stadd buffer in bytes. The buffer must be within the task's address space.

pn

Specifies one to four optional parameters that specify such additional information as block numbers for certain devices.

1.7.7 QIO$C IO.WLB—Writing a Logical Block

IO.WLB writes a block of data to the specified physical device unit.

Format

QIO$C IO.WLB,lun,[efn],[pri],[isb],[ast], <stadd,size,pn>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.

pri

Makes RSX-11M-PLUS and Micro/RSX QIO$ requests compatible with RSX-11D. Thus, a value of 0 (or a null) should be used for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Section 1.4.1.6.

ast

Specifies the address of a service routine to be entered when an asynchronous system trap (AST) occurs. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.

stadd

Specifies the starting address of the data buffer. The address must be word aligned for certain drivers; otherwise, stadd may be on a byte boundary.

size

Specifies the size of the stadd buffer in bytes. The buffer must be within the task's address space.

pn

Specifies one to four optional parameters that specify such additional information as block numbers or format control characters for certain devices.

1.7.8 QIO$C IO.WVB—Writing a Virtual Block

IO.WVB writes a virtual block of data to a physical device unit. A virtual block indicates a block position relative to the start of a file. For sequential, record-oriented devices such as terminals and line printers, the Executive converts IO.WVB to IO.WLB.

All tasks should use IO.WVB rather than IO.WLB to write for file-structured devices. However, if you issue a virtual write for a file-structured device (disk or DECtape II), you must ensure that a file is open on the specified physical device unit. For record-oriented devices, you should use IO.WLB.

Note that any subfunction bits specified in the IO.WVB request (for example, TF.CCO, TF.WAL, or TF.WBT) are stripped when the IO.WVB is converted to an IO.WLB.

Format

QIO$C IO.WVB,lun,[efn],[pri],[isb],[ast], <stadd,size,pn>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Sections 1.3 and 1.4.1.3.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Section 1.4.1.4.

pri

Makes RSX-11M-PLUS and Micro/RSX QIO$ requests compatible with RSX-11D. Thus, a value of 0 (or a null) should be used for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Section 1.4.1.6.

ast

Specifies the address of a service routine to be entered when an asynchronous system trap (AST) occurs. If you want to interrupt your task to execute special code upon completion of this I/O request, you may specify ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing. For more information refer to Sections 1.4.4 and 1.4.5.

stadd

Specifies the starting address of the data buffer. The address must be word aligned for certain drivers; otherwise, stadd may be on a byte boundary.

size

Specifies the size of the stadd buffer in bytes. The buffer must be within the task's address space.

pn

Specifies one to four optional parameters that specify such additional information as block numbers or format control characters for certain devices.

1.8 User-Mode Diagnostic Functions

The I/O function code subfunction bit, IQ.UMD, provides support for user-mode diagnostics. You can execute standard I/O functions such as Read Logical Block, Write Logical Block, Attach to Device, and Detach from Device as user-mode diagnostics. To perform a diagnostic function, you must specify in the QIO$ Directive Parameter Block (DPB) the logical OR of IQ.UMD and the function you want to perform. For example, to perform a diagnostic Read Logical Block operation, specify the following as the QIO$ directive:

```
QIO$C  IO.RLB!IQ.UMD,lun,...
```

Support for user-mode diagnostics is always present for RSX–11M–PLUS and Micro/RSX, but not all drivers support user-mode diagnostic functions. Unpredictable device and driver behavior results when you set the IQ.UMD subfunction bit in QIO$s that are directed to the device if it does not support user-mode diagnostics. You can avoid problems if you issue a Get LUN (GLUN$) macro and check the user-mode diagnostics bit before emitting the user-mode diagnostic QIO$.

For a device to support user-mode diagnostics, the DV.UMD bit in the Unit Control Block (UCB) must be set. DV.UMD is at offset U.CW1 in the UCB.

In addition to standard I/O functions, both operating systems provide the following device-dependent, user-mode diagnostic functions:

- Disk diagnostic functions, as follows:

  IO.HMS Home seek or recalibrate

  IO.BLS Block seek (explicit seek)

  IO.OFF Offset position

  IO.RDH Read disk header

  IO.WDH Write disk header

  IO.WCK Write-check

- DECtape diagnostic functions, as follows:

  IO.RNF Read block number forward

  IO.RNR Read block number reverse

- Magnetic tape diagnostic functions, as follows:

  IO.LPC Read longitudinal parity character

  IO.ERS Erase tape

UMDIO$ is the macro that defines these functions.

To execute a user-mode diagnostic function, you must first attach a device for diagnostics by using I/O function code IO.ATT!IQ.UMD. Execute the diagnostic functions and then detach the device.

The parameter list in words 1 to 6 of the DPB should contain the following information:

- I/O buffer address.
- I/O buffer size.
- Offset factor for disks with offset recovery. To determine the offset factor, refer to the offset register in the hardware reference manual; this parameter is not used if the device does not have offset recovery.
- Double-precision logical block number.
- Your task's register buffer address (the I/O driver copies its hardware registers to this buffer in your program); see a hardware reference manual for the length of the address.

A typical DPB for a diagnostic function might look like the following:

```
$DSKPB::
         .BYTE   3,12.              ; Size of the DPB, QIOW
                                    ;   directive code
         .WORD   IO.WDH!IQ.UMD      ; I/O function code
         .WORD   THELUN             ; Logical Unit Number
         .BYTE   THEEFN,0           ; Event flag number
         .WORD   $IOSTS             ; I/O status block address
         .WORD   0                  ; AST address

$IOBUF:: .WORD   0                  ; Buffer address
         .WORD   0                  ; Transfer size in bytes
         .WORD   0                  ; Device dependent
$LBH::   .WORD   0                  ; High-order logical block number
$LBL::   .WORD   0                  ; Low-order logical block number
         .WORD   $RGBUF             ; Register buffer address
```

The user-mode diagnostic functions return either Success (IS.SUC) or Device Not Ready (IE.DNR) messages. No other error codes are returned. All error recovery is completely up to you. Any errors that occur are not logged in the error log.

A typical program fragment, using the user-mode diagnostic functions, might look like the following:

```
         .MCALL  UMDIO$,ALUN$S,QIO$S
UMDIO$                              ; Define diagnostic functions
ALUN$S   #14.,#"DM,#0               ; Associate DM0 with lun 14
.
.
.
QIO$S    #IO.ATT!IQ.UMD,#14.        ; Attach DM for diagnostic I/O
.
.
.
QIO$S    #IO.RDH!IQ.UMD,#14.,,,,,<#$IOBUF,#512.,,#LBH,#LBL,#$RGBUF>
                                    ; Read disk header
.
.
.
```

```
QIO$S    #IO.RLB!IQ.UMD,#14.,,,,,<#$IOBUF,#512.,,#LBH,#LBL,#$RGBUF>
                                 ; Read logical block
.
.
.
QIO$S    #IO.DET!IQ.UMD,#14.      ; Detach DM
.
.
.
```

1.9 I/O Completion

When the system completes an I/O request, either successfully or unsuccessfully, the Executive selects return conditions depending upon the parameters included in the QIO$ macro call. There are three major returns:

- The Executive declares a significant event when an I/O operation completes execution. If you included an efn parameter in the I/O request, the corresponding event flag is set.
- If you included an isb parameter in the QIO$ macro call, the Executive returns a code identifying the type of success or failure. The code is in the low-order byte of the first word of the I/O status block (IOSB) at the location represented by isb.

  This status return code is of the form IS.xxx (success) or IE.xxx (error). For example, if the device accessed by the I/O request is not ready, a status code of IE.DNR is returned in isb. The section named Return Codes summarizes general codes returned by most of the drivers described in this manual.

  If the isb parameter was omitted, the requesting task cannot determine whether the I/O request was successfully completed. A carry clear return from the directive itself simply means that the directive was accepted and the I/O request was queued, not that the actual I/O operation was successfully performed.
- If you specified an ast parameter in the QIO$ macro call, a branch to the AST service routine, beginning at the location identified by ast, occurs when the I/O operation completes execution.

1.9.1 Return Codes

The Executive recognizes and handles the following two kinds of status conditions when they occur in I/O requests:

Directive	Indicate the acceptance or rejection of the QIO$ directive itself.
I/O status	Indicate the success or failure of the I/O operation.

Directive conditions relevant to I/O operations may indicate any of the following:

- Directive acceptance
- Invalid buffer specification
- Invalid efn parameter
- Invalid lun parameter
- Invalid Directive Identification Code (DIC) number or Directive Parameter Block (DPB) size

- Unassigned logical unit number (LUN)
- Insufficient memory

The Executive returns a code indicating the acceptance or rejection of a directive to the Directive Status Word (DSW) at symbolic location $DSW. You can test this location to determine the type of directive condition.

I/O conditions indicate the success or failure of the I/O operation that you specified in the QIO$ macro. I/O driver errors include such conditions as Device Not Ready, Privilege Violation, File Already Open, or Write-Locked Device. If you include an isb parameter in the QIO$ directive, identifying the address of a 2-word IOSB, the Executive returns an I/O status code in the low-order byte of the first word of this block when an I/O operation completes execution. This code is a binary value corresponding to a symbolic name of the form IS.xxx or IE.xxx. You can test the low-order byte of the word symbolically, by name, to determine the type of status return. The system object module library defines the correspondence between global symbolic names and directive and I/O completion status codes. You may also obtain local symbolic definitions by using the DRERR$ and IOERR$ macros, which reside in the System Macro Library and are summarized in Appendix B.

Binary values of status codes always have the following meanings:

Code	Meaning
Positive (greater than 0)	Successful completion
0	Operation still pending
Negative	Unsuccessful completion

A pending operation means that the I/O request is still in the queue of requests for the respective driver, or the driver has not yet completely serviced the request.

1.9.2 Directive Conditions

Table 1–4 summarizes the directive conditions that your task may encounter by issuing QIO$ directives. The table lists acceptance condition first, followed by error codes indicating various reasons for rejection.

Table 1–4: Directive Conditions

Code	Reason
IS.SUC	**Directive accepted** The first six parameters of the QIO$ directive were valid, and sufficient dynamic memory was available to allocate an I/O packet.
IE.ADP	**Invalid address** The IOSB or the QIO$ DPB was outside of the issuing task's address space or was not aligned on a word boundary.

Table 1–4 (Cont.): Directive Conditions

Code	Reason
IE.IEF	**Invalid event flag number** The efn specification in a QIO$ directive was less than 0 or greater than 96.
IE.ILU	**Invalid logical unit number** The lun specification in a QIO$ directive was invalid for the issuing task. For example, there were only five logical unit numbers associated with the task, and the value specified for lun was greater than 5.
IE.SDP	**Invalid DIC number or DPB size** The directive identification code (DIC) or the size of the Directive Parameter Block (DPB) was incorrect; the legal range for a DIC is from 1 to 127, and all DIC values must be odd. Each individual directive requires a DPB of a certain size. If the size is not correct for the particular directive, this code is returned. The size of the QIO$ DPB is always 12 words.
IE.ULN	**Unassigned LUN** The logical unit number in the QIO$ directive was not associated with a physical device unit. Your task may recover from this error by issuing a valid Assign LUN (ALUN$) directive and then by reissuing the rejected directive.
IE.UPN	**Insufficient dynamic memory** There was not enough dynamic memory to allocate an I/O packet for the I/O request. You can try again later by blocking the task with a WTSE$ macro. Note that WTSE$ is the only effective way for the issuing task to block its execution, because other directives usable for this purpose require dynamic memory for their execution (for example, Mark Time (MRKT$)).

1.9.3 I/O Status Conditions

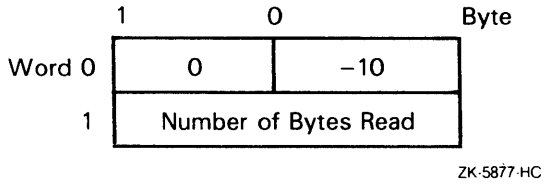
I/O status is returned in a 2-word IOSB upon completion of the I/O operation. The status may show a successful completion or an error. The contents of the 2-word IOSB is explained as follows:

- The low-order byte of the first word receives a status code of the form IS.xxx (success) or IE.xxx (error) when an I/O operation completes execution.
- The high-order byte of the first word is usually device dependent.
- The second word contains the number of bytes transferred or processed if the operation is successful and involves reading or writing.

If the isb parameter of the QIO$ directive is omitted, this information is not returned.

Figure 1–3 illustrates an example 2-word IOSB on completion of a terminal read operation.

Figure 1–3: I/O Status Block for Terminal Read Operation



In the figure, the number -10 is the status code for IE.EOF (end-of-file). If this code is returned, it indicates that input was terminated by pressing CTRL/Z, which is the end-of-file termination sequence on a terminal.

To test for a particular error condition, your task generally should compare the low-order byte of the first word of the IOSB with a symbolic value, as follows:

```
CMPB    #IE.DNR,IOSB
```

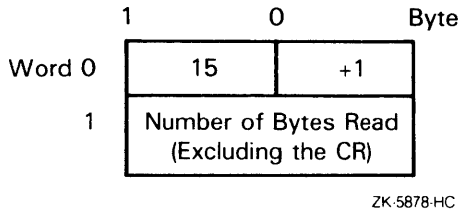
However, to test for certain types of successful completion of the I/O operation, the entire word value must be compared. For example, if a carriage return terminated a line of input from the terminal, a successful completion code of IS.CR is returned in the IOSB. If an Escape (or Altmode) character was the terminator, a code of IS.ESC is returned. To check for these codes, your task should first test the low-order byte of the first word of the block for IS.SUC and then test the full word for IS.CC, IS.CR, IS.ESC, or IS.ESQ. (Other success codes that must be read in this manner are listed in Appendix B).

Note that both of the following comparisons test as equal because the low-order byte in both cases is +1:

```
CMP     #IS.CR,IOSB
CMPB    #IS.SUC,IOSB
```

Figure 1–4 illustrates the status block of a successful completion where the carriage return is the terminal indicator (IS.CR).

Figure 1–4: I/O Status Block for IS.xxx



In the figure, 15 is the octal code for carriage return and +1 is the status code for successful completion.

Table 1–5 summarizes status codes that may be returned in the IOSB specified in the QIO$ directive on completion of the I/O request. The codes described in Table 1–5 are general status codes that apply to the majority of devices presented in subsequent chapters. Error codes specific to only one or two drivers are described only in relation to the devices for which they

are returned. Table 1–5 describes successful and pending codes first, and then describes error codes.

Table 1–5: I/O Status Conditions

Code	Reason
IS.SUC	**Successful completion** The I/O operation specified in the QIO$ directive was completed successfully. The second word of the IOSB can be examined to determine the number of bytes processed if the operation involved reading or writing.
IS.PND	**I/O request pending** The I/O operation specified in the QIO$ directive has not yet been executed. The I/O status block is filled with zeros.
IE.ABO	**Operation aborted** The specified I/O operation was canceled while in progress or while still in the I/O queue.
IE.ALN	**File already accessed on LUN** The task attempted to open a file on the physical device unit associated with the specified LUN, but a file has already been opened by the issuing task on that LUN.
IE.BAD	**Bad parameters** An invalid specification was supplied for one or more of the device-dependent QIO$ parameters (words 6–11). For example, a bad channel number or gain code was specified in an analog-to-digital (A/D) converter I/O operation.
IE.BBE	**Bad block on device** One or more bad blocks were found. Data cannot be written on or read from bad blocks.
IE.BLK	**Illegal block number** An invalid block number was specified for a file-structured physical device unit. This code is returned, for example, if block 4800 is specified for an RK05 disk on which legal block numbers extend from 0 to 4799.
IE.BYT	**Byte-aligned buffer specified** Byte alignment was specified for a buffer, but only word (or doubleword) alignment is legal for the physical device unit. For example, a disk function requiring word alignment was requested, but the buffer was aligned on a byte boundary. Alternatively, the length of a buffer was not an appropriate multiple of bytes. For example, all RP03 disk transfers must be an even multiple of 4 bytes.
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached to the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.

Table 1–5 (Cont.): I/O Status Conditions

Code	Reason
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached to the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	**Device not ready** The physical device unit specified in the QIO$ directive was not ready to perform the desired I/O operation. This code is often returned as the result of an interrupt timeout; that is, a reasonable amount of time has passed, and the physical device unit has not responded.
IE.EOF	**End-of-file encountered** An end-of-file mark, record, or control character was recognized on the input device.
IE.FHE	**Fatal hardware error** The controller is physically unable to reach the location where input/output is to be performed on the device. The operation cannot be completed.
IE.IFC	**Illegal function code** A function code that was invalid for the specified physical device unit was specified in an I/O request. This code is returned if the task attempts to execute an invalid function or if, for example, a read function is requested on an output-only device, such as the line printer.
IE.NLN	**No file accessed on LUN** The task tried to close a file on the physical device unit associated with the specified LUN, but no file was currently open on that LUN.
IE.NOD	**Insufficient buffer space** Dynamic storage space has been depleted, and not enough buffer space was available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for such an operation.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO$ directive was not on line. When the system was bootstrapped, a device check indicated that this physical device unit was not in the configuration.
IE.OVR	**Illegal read overlay request** A read overlay was requested and the physical device unit specified in the QIO$ directive was not the physical device unit from which the task was installed. The read overlay function can be executed only on the physical device unit from which the task image containing the overlays was installed.
IE.PRI	**Privilege violation** The task that issued a request was not privileged to execute that request. For example, for the UDC11 and LPS11 devices, a checkpointable task attempted to connect to interrupts or to execute a synchronous sampling function.

Table 1–5 (Cont.): I/O Status Conditions

Code	Reason
IE.SPC	**Illegal address space** The following conditions can cause this error: • The buffer that your task requested for a read or write operation was partially or totally outside the address space of your task. • You specified a byte count of 0. • You specified TF.XCC and AST2 in the same QIO$ request.
IE.VER	**Parity error on device** After the system attempted its standard number of retries after an error occurred, the operation still could not be completed. This code is returned in the case of parity, Cyclic Redundancy Check (CRC), or similar errors.
IE.WCK	**Write-check error** An error was detected during the check (read) following a write operation.
IE.WLK	**Write-locked device** The task attempted to write on a write-locked physical device unit.

1.10 Powerfail Recovery Procedures for Disks and Dectape

Powerfail recovery recommendations for various devices are included in the following chapters. For disks and DECtape, power recovery asynchronous system traps (ASTs) should be used. Before returning for normal I/O operations, the AST service routine should provide a sufficient time delay, for the disk to attain normal operating speed before actually attempting read and write operations.

If QIO$s are being used for disk or DECtape I/O operations during powerfail recovery, an IE.DNR error status may be returned if the device is not up to operating speed when the request is issued. When this error is returned, your task should wait for the device to attain operating speed and attempt the I/O operation again prior to reporting an error. For example, an RK05 disk may require approximately 1 minute to attain operating speed after a power failure.

1.11 RSX–11M–PLUS and Micro/RSX Devices

RSX–11M–PLUS and Micro/RSX support the devices listed in Table 1–6 except as indicated. Digital Equipment Corporation supplies drivers for each of these devices. However, you should refer to your Software Product Description (SPD) to determine whether a device is supported by your computer's operating system. Table 1–6 lists the physical name, the driver, and the device description.

1.12 RSX-11M-PLUS and Micro/RSX Devices

RSX-11M-PLUS and Micro/RSX support the devices listed in Table 1-6 except as indicated. Digital Equipment Corporation supplies drivers for each of these devices. However, you should refer to your Software Product Description (SPD) to determine whether a device is supported by your computer's operating system. Table 1-6 lists the physical name, the driver, and the device description.

Table 1-6: Devices Supported by RSX-11M-PLUS and Micro/RSX

Physical Name	Driver	Description	Supported by Micro/RSX
TERMINAL DEVICES:			
TT	TTDRV	ASR/KSR-33 and ASR/KSR-35 teletypewriters	No
TT	TTDRV	All terminals supported by RSX-11M-PLUS and Micro/RSX, including the LA-, LQP-, VT05, VT50, VT61-, VT100-, VT200-, and RT02-series terminals. See the Software Product Description for your system.	
TT	TTDRV	DH11 and DH11-DM11-BB asynchronous communication line interface multiplexer	No
TT	TTDRV	DHV11 asynchronous communication line interface multiplexer	Yes
TT	TTDRV	DHU11 asynchronous communication line interface multiplexer	No
TT	TTDRV	DL11-A, DL11-B, DL11-C, DL11-D, DL11-E and DL11-W asynchronous communication line interfaces	No
TT	TTDRV	DLV11-E, DLV11-F asynchronous communication line interfaces	Yes
TT	TTDRV	DZ11 asynchronous communication line interface multiplexer	No
TT	TTDRV	DZV11 asynchronous communication line interface multiplexer	Yes
TT	TTDRV	DZQ11 Q-Bus 4-line terminal multiplexer	
DISK DEVICES:			
DB	DBDRV	RP04/RP05/RP06 disk pack	No
DF	DFDRV	RF11/RS11 fixed-head disk	No
DK	DKDRV	RK11/RK05 or RK05F cartridge disk	No
DL	DLDRV	RLV12/RL01/RL02 cartridge disk	Yes

Table 1–6 (Cont.): Devices Supported by RSX–11M–PLUS and Micro/RSX

Physical Name	Driver	Description	Supported by Micro/RSX
DISK DEVICES:			
DM	DMDRV	RK611/RK06 or RK07 cartridge disk	No
DP	DPDRV	RP11/RP02 or RP03 disk pack	No
DR	DRDRV	RM02/RM03/RM05 disk pack	No
DR	DRDRV	RM80, RP07 fixed-media disk	No
DS	DSDRV	RS03/RS04 fixed-head disk	No
DU	DUDRV	KDA50/UDA50/RA80/RA81/RA82 fixed-media disk	Yes
DU	DUDRV	KDA50/UDA50/RA60 disk pack	Yes
DU	DUDRV	RC25 fixed-media and removable cartridge disk subsystem	Yes
DU	DUDRV	RD51/RD52/RD53/RD54 fixed-media disk	Yes
DU	DUDRV	RX50/RX33 flexible disk	Yes
DX	DXDRV	RX11/RX01 flexible disk	No
DY	DYDRV	RX211/RX02 flexible disk	Yes
EM	EMDRV	ML-11 fast electronic mass-storage device	No
TAPE DEVICES:			
DD	DDDRV	DL11/TU58 DECtape II	
MS	MSDRV	TU80 magnetic tape subsystem	No
MS	MSDRV	TSV05/TK25 magnetic tape subsystem	Yes
MS	MSDRV	TS11 magnetic tape subsystem	No
MT	MTDRV	TM11 magnetic tape controller with TE10, TU10, or TS03 drive	No
MM	MMDRV	RH11/70 controller with TM02/03 formatter and TE16, TU16, or TU45 drive	No
MM	MMDRV	RH11/70 controller with TM03 formatter and TU77 drive	No
MU	MUDRV	TK50 cartridge tape drive	Yes
MU	MUDRV	TU81 tape drive	No

Table 1–6 (Cont.): Devices Supported by RSX–11M–PLUS and Micro/RSX

Physical Name	Driver	Description	Supported by Micro/RSX
PRINTER DEVICES:			
LP	LPDRV	LP11 controller with LP14, LP01, LP02, LP04, LP05, LP06, LP07, LP26, and LP27 line printers	No
LP	LPDRV	LPV11/LP25/LP26 line printers, LN01/LN03 laser printer	Yes
LP	LPDRV	LS11 controller and line printer	No
LP	LPDRV	LV11 controller with LV01 line printer	No
LP	LPDRV	LA180 controller and line printer	No
CARD READER DEVICES:			
CR	CRDRV	CR11/CM11 card reader	No
COMMUNICATION LINE DEVICES:			
XB	XBDRV	DA11-B asynchronous communication line interface	No
XL	XLDRV	DL11-E asynchronous communication line interface	No
XL	XLDRV	DLV11-E asynchronous communication line interface	No
XC	XMDRV	DMC11 synchronous communication line interface	No
XE	XEDRV	RSX QIO DEUNA driver	No
XW	XWDRV	DUP11 synchronous communication line interface	No
LABORATORY DEVICE:			
LA	LADRV	LPA11-K Laboratory Peripheral Accelerator	No
PAPER TAPE DEVICES:			
PP	PPDRV	PC11 paper tape reader/punch	No
PR	PRDRV	PR11 paper tape reader	No

Table 1–6 (Cont.): Devices Supported by RSX–11M–PLUS and Micro/RSX

Physical Name	Driver	Description	Supported by Micro/RSX
NULL DEVICE:			
NL	NLDRV	Null device driver; a software construct to eliminate unwanted output	Yes
K-SERIES LABORATORY PERIPHERAL DEVICES:			
		AA11-K digital-to-analog converter and display	No
		AD11-K analog-to-digital converter	No
		AM11-K multiple-gain multiplexer	No
		DR11-K digital I/O interface	No
		KW11-K programmable real-time clock	No
COMMUNICATIONS DEVICES:			
LR/LT	LRDRV	PCL11 parallel communications link	No
LR/LT	LRDRV	PCL11-A/PCL11-B receiver port	No
OTHER DEVICES:			
QA-QZ	Any	A physical name reserved for customer use	Yes
JA-JZ	Any	A physical name reserved for customer use	Yes
ZA-ZZ	Any	A physical name reserved for customer use	Yes

Chapter 2
Full-Duplex Terminal Driver

2.1 Introduction to the Full-Duplex Terminal Driver

This chapter describes the use of the full-duplex terminal driver (TTDRV.TSK) supplied with the RSX–11M–PLUS and Micro/RSX system. This chapter contains descriptions of all the QIO$ functions that you can use to read from or write to a full-duplex terminal. Additionally, it contains a description of terminal subfunctions that are specific to terminal drivers and that modify the action of the QIO$ functions. You can combine the subfunctions in a logical OR with the QIO$ function. Specific programming circumstances are combined with the description of the QIO$ function where they apply.

Throughout the remainder of this chapter, references made to Monitor Console Routine (MCR) can generally be applied to other command line interpreters (for example, the DIGITAL Command Language (DCL)). In addition, the prompt displayed on a terminal in response to invoking a command line interpreter (CLI) is appropriate for the specific CLI in use. For example, when MCR is invoked, the MCR prompt is displayed as follows:

```
>
```

2.1.1 Full-Duplex Terminal Driver

The full-duplex terminal driver described in this chapter works with a wide variety of terminals. It contains the following features:

- Full-duplex operation
- Type-ahead buffering
- Eight-bit characters
- Detection of hard receive errors
- Increased byte transfer length (8128 bytes)
- Additional terminal characteristics
- Additional terminal types
- Optional timeout on solicited input

- Device-independent cursor control
- Redisplay of prompt buffer when CTRL/R or CTRL/U is pressed
- Automatic XOFF character generation when a read is completed while in half-duplex mode, if requested
- Autobaud speed detection
- Added hardware support

2.1.2 Terminals Supported by the Full-Duplex Terminal Driver

The full-duplex terminal driver supports a variety of terminal devices, as listed in Table 2–1. Table 2–2 describes standard terminal interfaces. Subsequent sections describe each device in greater detail.

Table 2–1: Supported Terminal Devices

Model	Hardcopy Terminal	Columns	Lines/ Screen[1]	Character Set	Baud Range	Uppercase Send	Lowercase Receive
ASR-33/35	No	72		64	110		
DTC01	No				9600		
KSR-33/35	No	72		64	110		
LA12	Yes	132		96	50–9600	Yes	Yes
LA100	Yes	132		96	110–9600	Yes	Yes
LA30-P	Yes	80		64	300		
LA30-S	Yes	80		64	110–300		
LA34	Yes	132		96	110–300	Yes	Yes
LA36	Yes	132		64-96	110–300	Yes	Yes[2]
LA38	Yes	132		96	110–300	Yes	Yes
LA50	Yes	80/96/132			110–4800		
LA75	Yes	80/96/132			110–4800		
LA120	Yes	132		96	50–9600	Yes	Yes
LA180S	Yes	132		96	300–9600		Yes
LA210	Yes	132		96	300–9600		Yes
LA2XX	Yes	132		96	300–9600		Yes

[1] Applies only to video terminals.

[2] Only for 96-character terminal. The terminal driver supports the terminal interfaces summarized in Table 2–2. These interfaces are described in greater detail in Section 2.17. Programming is identical for all interfaces.

Table 2–1 (Cont.): Supported Terminal Devices

Model	Hardcopy Terminal	Columns	Lines/ Screen[1]	Character Set	Baud Range	Uppercase Send	Lowercase Receive
LN03	Yes			*[3]	1200–19200		Yes
LN03 PLUS	Yes			*[3]	1200–19200		Yes
LQP02	Yes	132			110–9600		
LQP03	Yes	132/158			110-9600		
RT02	No	64	1	64	110–1200		
RT02-C	No	64	1	64	110–1200		
VT05B	No	72	20	64	110–2400	Yes	
VT50	No	80	12	64	110–9600		
VT50H	No	80	12	64	110–9600		
VT52	No	80	24	96	110–9600	Yes	Yes
VT55	No	80	24	96	110–9600	Yes	Yes
VT61	No	80	24	96	110–9600	Yes	Yes
VT100	No	80/132	24	96	50–9600	Yes	Yes
VT101	No	80/132	24	96	50–19200	Yes	Yes
VT102	No	80/132	24	96	50–9600	Yes	Yes
VT105	No	80/132	24	96	50–19200	Yes	Yes
VT125	No	80/132	24	96	50–9600	Yes	Yes
VT131	No	80/132	24	96	50–19200	Yes	Yes
VT132	No	80/132	24	96	50–19200	Yes	Yes
VT220	No	80/132	24	94[4]	50–19200	Yes	Yes
VT240	No	80/132	24	94[4]	50–19200	Yes	Yes
VT241	No	80/132	24	94[4]	50–19200	Yes	Yes

[1] Applies only to video terminals.

[3] Includes the DEC Multinational Character Set.

[4] Five character sets of 94 characters each. Includes the DEC Multinational Character Set.

Table 2–2: Standard Terminal Interfaces

Model	Type
DH11	16-line multiplexer[1]
DHU11	UNIBUS 16-line asynchronous multiplexer
DHV11	8-line multiplexer[2]
DH11-DM11-BB	16-line multiplexer with modem control[3]
DJ11	16-line multiplexer
DL11-A/B/C/D/E/W	Single-line interfaces
DLV11-E/F	Single-line interfaces[4]
DZ11	8-line multiplexer with modem control[4]
DZQ11	Q-bus 4-line terminal multiplexer
DZV11	Q-bus 4-line asynchronous multiplexer
CXA16	LSI-11/Q-bus 16-line BA200-series asynchronous multiplexer
CXB16	LSI-11/Q-bus 16-line BA200-series asynchronous multiplexer
CXY08	Q-bus 8-line BA200-series asynchronous multiplexer

[1] Direct memory access (DMA) is supported in the full-duplex terminal driver only.

[2] Full duplex terminal driver only.

[3] Full-duplex control only. For example, in the United States, a Bell 103A-type modem provides full-duplex control only.

[4] DLV11 support with modem control is provided in the full-duplex terminal driver only.

Terminal input lines can have a maximum length of 8128 (8K minus 64) bytes. Extra characters of an input line that exceed the maximum line length generally become an unsolicited input line if the terminal is not attached with the type-ahead buffering feature enabled. The full-duplex terminal driver discards all unsolicited input from an unattached, slave terminal.

2.1.2.1 ASR-33/35 Teletypewriters

The ASR-33 and ASR-35 teletypewriters are asynchronous, hardcopy terminals. No paper-tape reader or punch capability is supported.

2.1.2.2 KSR-33/35 Teletypewriters

The KSR-33 and KSR-35 teletypewriters are asynchronous, hardcopy terminals.

2.1.2.3 LA12 Portable Terminal

The LA12 is a personal, portable, hardcopy terminal.

2.1.2.4 LA100 DECprinter

The LA100 is a desktop, matrix, hardcopy terminal.

2.1.2.5 LA30 DECwriter

The LA30 DECwriter is an asynchronous, hardcopy terminal that is capable of producing an original and one copy. The LA30-P is connected by a parallel line and the LA30-S is connected by a serial line.

2.1.2.6 LA36 DECwriter

The LA36 DECwriter is an asynchronous terminal that produces hard copy and operates in serial mode. It has an impact printer capable of generating multipart and special preprinted forms. The LA36 can receive and transmit both uppercase and lowercase letters.

2.1.2.7 LA34/38 DECwriters

The LA34 DECwriter is an asynchronous terminal that produces hard copy and uses a platen paper-feed mechanism.

The LA38 DECwriter includes a detachable tractor-feed mechanism for use with continuous forms.

2.1.2.8 LA120 DECwriter

The LA120 DECwriter is a hardcopy, uppercase and lowercase terminal. It can print multipart forms at speeds up to 180 characters per second. You can select serial communications speed from 14 baud rates ranging from 50 to 9600 bits per second (bps) the terminal driver supports split transmit and receive baud rates. Hardware features allow bidirectional printing for maximum printing speed, and they also allow you to select features, including font size, line spacing, tabs, margins, and forms control. Also, you can set up these functions if you issue appropriate ANSI-standard escape sequences.

2.1.2.9 LA180S DECprinter

The LA180S DECprinter is a serial version of the LA180. It is a print-only device (it has no keyboard) that can generate multipart forms. The LA180S can print uppercase and lowercase letters.

2.1.2.10 LQP02 Letter-Quality Printer

The LQP02 letter-quality printer is a formed-character, desktop printer incorporating daisywheel technology. This letter-quality printer offers over 100 character sets and handles regular office stationery up to a maximum of 15 inches (with a print capacity of 13.5 inches). You can select lines per inch and characters per inch: 10 or 12 characters per inch and 2, 3, 4, 6, and 8 lines per inch. At 10 characters per inch you get 132 columns, and at 12 characters per inch you get 158 columns. The buffer capacity is 256_{10} characters.

2.1.2.11 LQP03 Letter-Quality Printer

The LQP03 letter-quality printer is a compact, 130-petal daisywheel printer that prints on multipart forms and fanfold paper. It prints at speeds of 25 characters per second maximum in 10-pitch Shannon text and 34 characters per second in 12-pitch triple-A text. It also includes support for all the same features as the LQP02.

2.1.2.12 LA50 Personal Printer

The LA50 personal printer is a desktop dot-matrix impact printer. It has two print modes: text mode and enhanced print mode. In text mode, it prints 100 characters per second. In enhanced print quality mode, it prints 50 characters per second and creates a crisper, more uniform character than an ordinary dot-matrix printer. You can choose 10, 12, or 16 characters per inch that print up to 80, 96, or 132 columns respectively. There can be 6, 8, or 12 lines per inch. The buffer capacity is 255_{10} characters.

2.1.2.13 LA75 Personal Printer

The LA75 personal printer is a versatile and reliable high-speed, dot-matrix impact printer. It includes all the features of the LA50. In addition, the LA75 contains a built-in tractor feed that accepts form-fed paper and labels. The LA75 allows single-sheet printing and can print text in five modes: draft, memo, near letter-quality, letter quality, and bit-mapped graphics. Additionally, it has a font cartridge capability that provides plug-in Letter Gothic or Orator fonts for meeting special printing requirements.

2.1.2.14 LA210 Letter Printer

The LA210 letter printer is a desktop, dot-matrix tex and graphics printer. It can print text in four modes: draft, memo, near letter-quality, and full bit-mapped graphics. The LA210 can print on office stationery, multipart forms, labels, roll and fanfold paper up to 14.9 inches wide. It supports numerous standard resident typefaces, and includes optional font cartridges and ROM chips.

2.1.2.15 LN03/LN03 PLUS Laser Printers

The LN03 laser printers are desktop units that employ electrophotographic imaging and xerographic printing. They print at a speed of 8 pages per minute. The print resolution is 300 by 300 dots per inch. They contain up to 17 resident fonts in three typefaces, including ASCII, multinational, and technical character sets. Both printers print in portrait and landscape mode.

The LN03 PLUS also includes the following additional features:

- Full-page, bit-mapped graphics that are compatible with Digital's sixels and Tektronix[1] 4010/4014 graphics protocols
- One megabyte (Mb) of on-board RAM for text/graphics applications

[1] Tektronix is a registered trademark of Tektronix, Inc.

2.1.2.16 RT02 Alphanumeric Display Terminal and RT02-C Badge Reader/Alphanumeric Display Terminal

The RT02 is an alphanumeric display terminal for applications in which source data is primarily numeric. A shift key permits the entry of 30 discrete characters, including uppercase alphabetic characters. The RT02 can, however, receive and display 64 characters.

The RT02-C model also contains a badge reader. This option provides a reliable method of identifying and controlling access to the PDP–11 minicomputer or to a secure facility. Furthermore, data in a format corresponding to that of a badge (22-column fixed data) can be entered quickly.

2.1.2.17 VT05B Alphanumeric Display Terminal

The VT05B is an alphanumeric display terminal that consists of a cathode-ray tube (CRT) display and a self-contained keyboard. The VT05B offers direct cursor addressing.

2.1.2.18 VT50 Alphanumeric Display Terminal

The VT50 is an alphanumeric display terminal that consists of a CRT display and a keyboard. It is similar to the VT05B in operation, but it does not offer direct cursor addressing.

2.1.2.19 VT50H Alphanumeric Display Terminal

The VT50H is an alphanumeric display terminal with CRT display, keyboard, and numeric keypad. It offers direct cursor addressing, but its direct cursor addressing is not compatible with that of the VT05B.

2.1.2.20 VT52 Alphanumeric Display Terminal

The VT52 is an uppercase and lowercase alphanumeric terminal with CRT display. It also has a numeric keypad and direct cursor addressing. The VT52's direct cursor addressing is compatible with that of the VT50H, but not with that of the VT05B. The VT52 can be configured with a built-in thermal printer.

2.1.2.21 VT55 Graphics Display Terminal

The VT55 is similar to the VT52 in its operation as an alphanumeric terminal. The VT55 offers graphics display features that are accessible by a task.

2.1.2.22 VT61 Alphanumeric Display Terminal

The VT61 is an uppercase and lowercase alphanumeric terminal with an integral microprocessor. It offers two 128-member character sets and numerous built-in functions for editing and forms preparation as well as a block-transfer mode.

2.1.2.23 VT100 Terminal

The VT100 terminal is an uppercase and lowercase alphanumeric keyboard and video display terminal. It can display 24 lines of 80 to 132 characters per line. You can select serial communications speed from baud rates ranging from 50 to 9600 bps. Hardware features allow you to select display characteristics and functions including smooth scroll, reverse video, and so forth. The system also sets up these functions if you issue appropriate ANSI-standard escape sequences.

2.1.2.24 VT101 Terminal

The VT101 terminal is functionally identical to the VT100. However, it does not support the advanced video features.

2.1.2.25 VT102 Terminal

The VT102 terminal is functionally identical to the VT100. However, it does not have any expansion capability. It has enhanced modem control, supports advanced video features, and includes a port for a printer.

2.1.2.26 VT105 Terminal

The VT105 terminal is an alphanumeric and graphics display video terminal. The VT105 can display two graphs, two shaded graphs, or two strip charts. These graphs may have alphanumeric labels.

2.1.2.27 VT131 Terminal

The VT131 is the same as the VT102 with the addition of built-in editing features.

2.1.2.28 VT220 Terminal

The VT220 terminal is a general-purpose video display terminal displaying 24 rows of 80 or 132 columns. It has: ANSI-compatible control functions; user-definable function keys; video reverse, bold, underline, blink, double height/double width line attributes; and can run in VT100, VT200 7-bit, VT200 8-bit, and VT52 modes. Setup state allows you to configure the terminal and examine its status.

2.1.2.29 VT240 Terminal

The VT240 terminal is a general-purpose video display terminal displaying 24 rows of 80 or 132 columns. It has: ANSI-compatible control functions; user-definable function keys; and video reverse, bold, underline, blink, double height/double width line attributes. It can run in VT100, VT200 7-bit, VT200 8-bit, VT52, Tektronix 4010/4014, and ReGIS graphics modes. Setup state allows you to configure the terminal and examine its status. The VT240 has graphics capability to draw points, vectors, circles, arcs, and curves.

2.1.2.30 VT241 Terminal

The VT241 terminal is functionally identical to the VT240 terminal except that the VT241 has a color monitor.

2.2 Get LUN Information Macro

The Get LUN information directive (GLUN$) instructs the system to fill a 6-word buffer with information about the physical device unit to which the LUN is assigned. For more information about this directive, refer to Get LUN in the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual.* The following section describes the information that Get LUN makes available for terminals in word 2 of the buffer.

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the terminal information shown in Table 2-3. A setting of 1 indicates that the described characteristic is true for terminals.

Table 2-3: Word 2 of the Get LUN Macro Buffer

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	1	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Mass-storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device is mountable as a communications channel
14	0	Device is mountable as a Files-11 Volume
15	0	Device is mountable

Words 3 and 4 of the buffer are undefined. Word 5 indicates the default buffer size (the width of the terminal carriage or display screen).

2.3 QIO$ Macro

Standard QIO$ functions may be used with any device; whereas, device-specific QIO$ functions apply only to specific devices or uses.

2.3.1 Format of QIO$C for Standard Functions

The QIO$ macros formats for standard functions are shown next.

Formats

```
QIO$C   { IO.ATT }
        { IO.DET } ,...,
        { IO.KIL }

QIO$C   { IO.RLB }
        { IO.RVB } ,...,  <stadd,size,[tmo]>

QIO$C   { IO.WLB }
        { IO.WVB } ,...,  <stadd,size,vfc>
```

2.3.2 Format of QIO$C for Device-Specific Functions

The QIO$ macro formats for device-specific functions are shown next.

Formats

```
QIO$C   IO.ATA,...,  <[ast1],[parameter2],[ast2]>

QIO$C   IO.CCO,...,  <stadd,size,vfc>

QIO$C   IO.EIO,...,  <stadd,size>

QIO$C   IO.HNG,...,

QIO$C   { IO.RAL }
        { IO.RNE } ,...,  <stadd,size,[tmo]>

QIO$C   IO.RPR,...,  <stadd,size,[tmo],pradd,prsize,vfc>

QIO$C   IO.RST,...,  <stadd,size,[tmo]>

QIO$C   IO.RTT,...,  <stadd,size,[tmo],table>

QIO$C   { IO.WAL }
        { IO.WBT } ,...,  <stadd,size,vfc>

QIO$C   { SF.GMC }
        { IO.GTS } ,...,  <stadd,size>

QIO$C   SF.SMC,...,  <stadd,size>
```

Table 2–4 lists the standard and device-specific functions of the QIO macro that are valid for terminals. The standard functions are described in Chapter 1. Some device-specific functions are options that may be selected during system generation. Two device-specific functions, SF.SMC and SF.GMC, have nonstandard function names.

Table 2–4: Standard and Device-Specific QIO Functions for Terminals

Format	Function
STANDARD FUNCTIONS:	
READ FUNCTIONS	
QIO$C IO.RLB,..., <stadd,size,[tmo]>	Read logical block (read typed input into buffer).
QIO$C IO.RVB,..., <stadd,size,[tmo]>	Read virtual block (read typed input into buffer).
WRITE FUNCTIONS	
QIO$C IO.WLB,..., <stadd,size,vfc>	Write logical block (print buffer contents).
QIO$C IO.WVB,..., <stadd,size,vfc>	Write virtual block (print buffer contents).
ATTACH, DETACH, AND CANCEL FUNCTIONS	
QIO$C IO.ATT,...	Attach device.
QIO$C IO.DET,...	Detach device.
QIO$C IO.KIL,...	Cancel I/O requests.
DEVICE-SPECIFIC FUNCTIONS:	
READ FUNCTIONS	
QIO$C IO.RAL,..., <stadd,size,[tmo]>	Read logical block; pass all characters.
QIO$C IO.RNE,..., <stadd,size,[tmo]>	Read logical block; do not echo.
QIO$C IO.RPR,..., <stadd,size,[tmo], pradd,prsize,vfc>	Read logical block after prompt.
QIO$C IO.RST,..., <stadd,size,[tmo]>	Read logical block ended by special terminators.
QIO$C IO.RTT,..., <stadd,size,[tmo],table>	Read logical block ended by specified special terminators.
WRITE FUNCTIONS	
QIO$C IO.WAL,..., <stadd,size,vfc>	Write logical block; pass all characters.
QIO$C IO.WBT,..., <stadd,size,vfc>	Write logical block; breakthrough I/O conditions at terminal.

Table 2–4 (Cont.): Standard and Device-Specific QIO Functions for Terminals

Format	Function
DEVICE-SPECIFIC FUNCTIONS:	
MISCELLANEOUS FUNCTIONS	
QIO$C IO.ATA,..., <ast,[parameter2],[ast2]>	Attach device, specify unsolicited input-character asynchronous system trap (AST).
QIO$C IO.CCO,..., <stadd,size,vfc>	Cancel CTRL/O (if in effect), then write logical block.
QIO$C IO.EIO { !TF.RLB / !TF.WLB },..., <stadd,size>	Extended I/O.
QIO$C IO.GTS,..., <stadd,size>	Get terminal support.
QIO$C IO.HNG,...	Hang up remote line.
QIO$C SF.GMC,..., <stadd,size>	Get multiple characteristics.
QIO$C SF.SMC,..., <stadd,size>	Set multiple characteristics.

2.3.3 Parameters

The parameters for the various QIO$ macros and their descriptions are shown next.

Parameters

ast

Specifies the entry point for an unsolicited input-character AST.

ast2

Specifies the entry point for a CTRL/C AST.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

isb

Specifies the the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

parameter2

Specifies a number that you can specify in your task to identify this general terminal as the input source when an unsolicited character AST routine is entered.

pradd

Specifies the starting address of the byte buffer where the prompt is stored.

pri

Makes this QIO$ macro compatible with RSX–11D. Use a value of 0 or a null for this parameter.

prsize

Specifies the size of the pradd prompt buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

size

Specifies the size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space. For IO.EIO, SF.GMC, IO.GTS, and SF.SMC functions, size must be an even value.

stadd

Specifies the starting address of the data buffer. The address must be word aligned for IO.EIO, SF.GMC, IO.GTS, and SF.SMC; otherwise, stadd may be on a byte boundary.

table

Specifies the address of the 16-word, user-defined terminator table.

tmo

Specifies an optional timeout count specified in 10-second intervals. (For IO.EIO, the interval is specified in seconds.) Timeout is the maximum time allowed between two input characters before the read is aborted. The maximum timeout value is 255_{10} intervals.

If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer.

If you need more than 255_{10} intervals (or 255_{10} seconds for IO.EIO), issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.

vfc

Specifies cursor position.

If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter.

Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

2.3.4 Subfunction Bits

The terminal-specific functions described in this section are selected by using subfunction bits. A subfunction bit further modifies the action of an I/O function. A subfunction bit is specified by the name TF.xxx, and an I/O function is specified by the name IO.xxx, where xxx in each case is an acronym that represents the specific kind of function requested.

As an example, a QIO$ function to a terminal to request a read with no echo (IO.RNE) can be modified to read all characters. The "read all characters" subfunction bit is TF.RAL. To modify the function, you perform a logical OR of the subfunction bit with the QIO$ function in the QIO$ statement. To create the logical OR of the bit and the function, in this example, the QIO$ statement would appear as follows:

```
QIO$ IO.RNE!TF.RAL,....,...
```

See Section 2.4.2 for a listing of QIO$ functions and relative subfunction bits that can be issued.

Table 2–5 lists each subfunction bit with its symbolic name and meaning.

Table 2–5: Terminal Driver Subfunction Bits

Subfunction	Meaning
TF.AST	**Unsolicited input-character AST** For IO.ATT or IO.ATT!TF.ESQ, ast in the QIO$ macro specifies the address of an asynchronous system trap (AST) service routine to be entered when an unsolicited input character is entered. Control passes to ast whenever an unsolicited character (other than CTRL/Q, CTRL/S, CTRL/X, or CTRL/O) is entered at the terminal.
TF.BIN	**Binary prompt (send prompt as pass all)** The prompt is sent to the terminal without interpretation by the driver. This is similar, for the prompt, to a write-pass-all operation.
TF.CCO	**Cancel CTRL/O** The driver writes a logical block of data to the terminal regardless of a CTRL/O condition that may be in effect. If CTRL/O is in effect, it is canceled before the write occurs.
TF.ESQ	**Recognize escape sequences** Escape sequences recognition from the terminal are returned to the task. Otherwise, ESC is a line terminator. The subfunction TF.ESQ is for use with IO.ATA or IO.ATT!TF.AST.

Table 2–5 (Cont.): Terminal Driver Subfunction Bits

Subfunction	Meaning
TF.NOT	**Notification of unsolicited input** Unsolicited notification input causes an AST and entry into the AST service routine in the task. When the full-duplex terminal driver receives unsolicited terminal input (except CTRL/C) and you used the TF.NOT unsolicited input subfunction with IO.ATA, the resulting AST serves only as notification of unsolicited terminal input; the terminal driver does not pass the character to the task. Upon entry to the AST service routine, the high byte of the first word on the stack identifies the terminal causing the AST (parameter2) in the IO.ATA function. Using the TF.NOT subfunction allows a task to monitor more than one terminal for unsolicited input without continuously reading each terminal for possible unsolicited input. Note that the TF.NOT subfunction cannot be used with the CTRL/C AST (ast2) in unsolicited input IO.ATA); an unsolicited CTRL/C character flushes the type-ahead buffer.
TF.RAL	**Read all characters (pass all)** This subfunction allows the passage of all characters to the requesting task. The driver does not intercept control characters. The characteristic TC.8BC, when set, allows the driver to pass 8 bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and are not interpreted by the driver.
TF.RCU	**Restore cursor position** When defining cursor position in a function, you can use the TF.RCU subfunction to save the current cursor position. TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.
TF.RDI	**Read with default input** The default input that you specified in the extended I/O item list is displayed as an input line at the start of the read on the terminal. You may change this line or use it as input to the system. This subfunction is for use with the extended I/O function (IO.EIO) only.
TF.RES	**Read with escape sequence processing enabled** This subfunction enables escape sequence recognition for the read operation in extended I/O; it is effective for only one read.
TF.RLB	**Read logical block** This subfunction causes the driver to read a logical block from the specified terminal; it is for use with the extended I/O (IO.EIO) function only.
TF.RLU	**Read with lowercase-to-uppercase conversion** The task that uses this subfunction gets input in the buffer in uppercase; it is for use with the extended I/O (IO.EIO) function only.

Table 2–5 (Cont.): Terminal Driver Subfunction Bits

Subfunction	Meaning
TF.RNE	**Read with no echo** This subfunction reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information. CTRL/R is ignored while read with no echo is in progress.
TF.RNF	**Read with no filter** This subfunction reads and passes through CTRL/U, CTRL/R, and DELETE characters as normal characters. It is for use with the extended I/O (IO.EIO) function only.
TF.RPR	**Read after prompt** This subfunction is for use with the extended I/O only. The TF.RPR subfunction causes the driver to send a prompt to the terminal, and the driver immediately follows the prompt with a read function at the terminal. The TF.RPR acts as an IO.WLB (to write a prompt to the terminal) followed by IO.RLB. However, TF.RPR differs from the combination of those two functions as follows: • System overhead is lower with the TF.RPR because only one QIO$ is processed. • When using the TF.RPR function, there is no "window" during which a response to the prompt may be ignored. Such a window occurs if the task uses IO.WLB followed by an IO.RLB because no read may be posted at the time the response is received. • If the issuing task is checkpointable, it can be checkpointed during both the prompt and the read requested by the TF.RPR. • A CTRL/O that may be in effect prior to issuing the TF.RPR is canceled before the prompt is written. **Note** If a TF.RPR function is in progress when the driver receives a CTRL/R or CTRL/U, the prompt is redisplayed.
TF.RPT	**Read in pass-through mode** This subfunction passes all characters except XON/XOFF. It allows the passage of all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass 8 bits instead of 7. The driver intercepts the control characters CTRL/S and CTRL/Q. Other control characters, for example, CTRL/C, CTRL/O, and CTRL/Z, are passed to the task and are not interpreted by the driver. This subfunction is for use with the extended I/O (IO.EIO) function only.

Table 2–5 (Cont.): Terminal Driver Subfunction Bits

Subfunction	Meaning
TF.RST	**Read with special terminators** Special characters in the ranges $0–037_8$ and $175–177_8$ terminate the read. The driver does not interpret the terminating character. For example, a DELETE or RUBOUT 177_8 does not erase, and a CTRL/C does not produce a CLI prompt or, if CTRL/C abort is enabled, abort tasks. CTRL/U and CTRL/R do not perform their usual functions either. All control characters are terminators. TF.RST sets TF.TNE by default, which means that terminators are not echoed on the terminal screen. If uppercase-to-lowercase conversion is disabled, characters 175_8 and 176_8 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017_8, 021_8, and 023_8 respectively) are not special terminators. The driver interprets them as output control characters in a normal manner.
TF.RTT	**Read with terminator table** This subfunction is for use with the IO.EIO extended I/O function only. Control characters function normally with TF.RTT. Terminators echo by default. The additional use of subfunction TF.TNE prevents the echoing of terminators on the terminal screen. If you want to use special control characters as terminators, their normal function should be disabled with the subfunction TF.RNF or TF.RAL, or the characteristic TC.PTH. The terminator table (a bit mask table) length can be from $1–32_{10}$ bytes, where bit 0 is a null character, bit 1 is a CTRL/A, and so forth. The terminator table address is in the item list of the IO.EIO function. To use ASCII characters $128_{10}–255_{10}$, the characteristic TC.8BC must be set.
TF.TMO	**Read with timeout** This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the timeout count in 10-second intervals. (For IO.EIO, the interval is specified in seconds.) Timeout is the maximum time allowed between two input characters before the read is aborted. The maximum timeout value is 25_{10} intervals. If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer. If you need more than 25_{10} intervals (or 25_{10} seconds for IO.EIO), issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.
TF.TNE	**Read terminators with no echo** This subfunction allows for reading terminator characters from the terminal without their being echoed on the terminal screen as they are entered. It is for use with the extended I/O function (IO.EIO) only.

Table 2–5 (Cont.): Terminal Driver Subfunction Bits

Subfunction	Meaning
TF.WAL	**Write all characters** During a write-pass-all operation (as in IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation. It does not intercept control characters, and it does not keep track of cursor position. Long lines are not wrapped around if I/O wraparound has been selected.
TF.WBT	**Breakthrough write** This subfunction instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If another write function is currently in progress, it finishes the current request and the breakthrough write is the next write issued. Therefore, the TF.WBT subfunction cannot break through another breakthrough write that is in progress. The effect of this is that a CTRL/S can stop breakthrough write functions. Thus, it may be desirable for tasks to time out on breakthrough operations. If a read is currently posted, the breakthrough write proceeds, and an automatic CTRL/R redisplays any input that was received before the breakthrough write was effected (if the terminal is not in the full-duplex mode). CTRL/O, if in effect, is canceled. An escape sequence that was interrupted is rubbed out. Privileged tasks may issue a breakthrough write to any terminal. In addition, a nonprivileged task may issue a breakthrough write to the task's terminal. (The privileged MCR command BRO (broadcast) uses IO.WBT.)
TF.WIR	**Write with input redisplayed** This subfunction performs a write to the terminal. If a read is in progress at the terminal and you have entered characters in the input line, the prompt and the characters are redisplayed at the end of the write.
TF.WLB	**Write logical block to the specified device unit** Write logical block to the specified terminal. This subfunction is used with the extended I/O (IO.EIO) function only.

Table 2–5 (Cont.): Terminal Driver Subfunction Bits

Subfunction	Meaning
TF.XCC	**Exclude CTRL/C or abort active tasks** For use with the IO.ATA function. When TF.XCC is included in the IO.ATA function, all characters (except CTRL/C) are handled in the manner previously described. CTRL/C marks the beginning of a command line interpreter (CLI) line that is processed by a CLI task or, if CTRL/C abort is enabled, that aborts tasks active at the terminal. None of the characters, including the CTRL/C, are sent to the task issuing the function. Note that you can use either ast2 or TF.XCC, but not both, in the same QIO request. If both are specified in the request, an IE.SPC error is returned.
TF.XOF	**Send XOFF** The driver sends an XOFF to the terminal after it is read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

See Section 2.4.2 for a list of bits that can be combined in a logical OR with QIO$ functions. If a task invokes a subfunction bit that is not supported on the system, the subfunction bit is ignored, and the QIO$ request is not rejected. For example, if breakthrough write (TF.WBT) is not supported, an IO.WBT or IO.WLB!TF.WBT function is interpreted as an IO.WLB function.

In the following example, the QIO$ request uses more than one subfunction bit: a nonechoed read (TF.RNE), which is terminated by a special terminator character (TF.RST), and is preceded by a prompt:

```
QIO$C IO.RPR!TF.RNE!TF.RST,...,<stadd,size,,pradd,prsize,vfc>
```

2.4 Device-Specific QIO$ Functions

The following sections describe the device-specific functions for the full-duplex terminal driver. Some full-duplex terminal driver functions and features are system generation options. These options are briefly described in the following section.

2.4.1 System Generation Options in the Full-Duplex Terminal Driver

Following is a list of device-specific functions that are always included for the full-duplex terminal driver during system generation:

Unsolicited input character AST	Specifies an AST entry point for unsolicited input-character handling. This support is automatically included if your Executive supports asynchronous system traps (ASTs).
Breakthrough write	Instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If another write function is currently in progress, it finishes the current request and the breakthrough write is the next write issued. Therefore, the TF.WBT subfunction cannot break through another breakthrough write that is in progress. The effect of this is that a CTRL/S can stop breakthrough write functions. Thus, it may be desirable for tasks to time out on breakthrough operations. If a read is currently posted, the breakthrough write proceeds, and an automatic CTRL/R is performed to redisplay any input that was received before the breakthrough write was effected (if the terminal is not in the full-duplex mode). CTRL/O, if in effect, is canceled. An escape sequence that was interrupted is rubbed out. Privileged tasks may issue a breakthrough write to any terminal. In addition, a nonprivileged task may issue a breakthrough write to the task's terminal. (The privileged MCR command BRO (broadcast) uses IO.WBT.)
CTRL/R retype	Sends a carriage return and line feed followed by the input buffer contents to the terminal whenever you press CTRL/R.
Escape sequence handling	Recognizes and treats escape sequences as line terminators for all solicited input except read-pass-all requests. See the QIO$ functions IO.RAL, IO.RST, and IO.RTT in the following sections, and see Section 2.7 for a description of escape sequences.
Get Multiple Characteristics	Allows a task to determine the characteristics of individual terminals. See Section 2.4.15 for information about the QIO$ SF.GMC function.
Set Multiple Characteristics	Allows a task to set the physical characteristics of a terminal. See Section 2.4.16 for information about the QIO$ SF.SMC function.
Get Terminal Support	Allows a task to determine which terminal driver options were selected during system generation. See Section 2.4.6 for information about the QIO$ IO.GTS function.
Read After Prompt	Writes a prompt to the terminal and immediately follows it with a read. Reduces overhead and allows a task exclusive access to the terminal for the write and following read. See the QIO$ IO.RPR function in Section 2.4.10.

CRT Rubout	Allows the DELETE (or RUBOUT) key to erase a character from the CRT screen by echoing the characters to be deleted as backspace-space-backspace. See Section 2.6.2.
Hard Receive Error Detection	Known as Unrecoverable Input Error Notification in system generation. The driver flags framing errors, character parity errors, and data overruns and then passes the input characters to the requesting task with notification of an input error (including type). See Section 2.10.
Terminal-Independent Cursor Control	Allows the driver to output a cursor-positioning command before it outputs the contents of the buffer if you specify the vfc parameter for an output buffer.

The following device-specific functions are system generation options for the full-duplex driver:

Unsolicited input timeout	Discards unsolicited input when the timeout value that you specified during system generation expires.
Extended I/O	Allows the use of IO.EIO with TF.WLB or TF.RLB to increase the number of allowable I/O subfunctions.
Extended Network Command Terminal (NCT) Support	Allows the use of a terminal as a Network Command Terminal.
Modem Control	Sets the default answer speed for modems during system generation time but allows the speed to be changed on line with the SET command.

2.4.2 Functions and Allowed Subfunctions

Table 2–6 lists the functions with their allowed subfunctions. The subfunction bits are specified in the following QIO$C function descriptions; subfunction bits are described in general in Section 2.3.4.

Table 2–6: Summary of Subfunction Bits

Function	Equivalent Subfunctions	Allowed Subfunctions
STANDARD FUNCTIONS		
IO.ATT	None	TF.AST, TF.ESQ
IO.DET	None	None
IO.KIL	None	None
IO.RLB	None	TF.RAL, TF.RNE, TF.RST, TF.TMO, TF.XOF
IO.RVB[1]	None	TF.RAL, TF.RNE, TF.RST, TF.TMO, TF.XOF
IO.WLB	None	TF.CCO, TF.RCU, TF.WBT, TF.WAL
IO.WVB[1]	None	TF.CCO, TF.RCU, TF.WAL, TF.WBT

[1] Sufunctions are stripped off if they are specified with IO.RVB or IO.WVB.

Table 2-6 (Cont.): Summary of Subfunction Bits

Function	Equivalent Subfunctions	Allowed Subfunctions
DEVICE-SPECIFIC FUNCTIONS		
IO.ATA	IO.ATT!TF.AST	TF.ESQ, TF.NOT, TF.XCC
IO.CCO	IO.WLB!TF.CCO	TF.WAL, TF.WBT
IO.EIO[2]		TF.RLB, TF.WLB
SF.GMC		
IO.GTS		
IO.RAL	IO.RLB!TF.RAL	TF.RNE, TF.RST, TF.TMO, TF.XOF
IO.RNE	IO.RLB!TF.RNE	TF.RAL, TF.RST, TF.TMO, TF.XOF
IO.RPR		TF.BIN, TF.RAL, TF.RNE, TF.RST, TF.TMO, TF.XOF
IO.RST	IO.RLB!TF.RST	TF.RAL, TF.RNE, TF.TMO, TF.XOF
IO.RTT		TF.RAL, TF.RCU, TF.RNE, TF.TMO
SF.SMC		
IO.WAL	IO.WLB!TF.WAL	TF.CCO, TF.RCU, TF.WBT
IO.WBT	IO.WLB!TF.WBT	TF.CCO, TF.RCU, TF.WAL

[2]You must use TF.RLB or TF.WLB with IO.EIO, but you should not use both.

In addition to the device-specific QIO functions, the following sections also describe the use of subfunction bits.

2.4.3 QIO$C IO.ATA—Attach a Terminal with ASTs

The QIO$ IO.ATA macro attaches the terminal and identifies ast1 and ast2 as entry points for unsolicited input-character ASTs. With ast1 and ast2, IO.ATA specifies asynchronous system traps (ASTs) to process unsolicited input characters entered at the terminal. A minimum of one AST parameter (ast or ast2) is required.

IO.ATA is equivalent to the IO.ATT attach function executed in a logical OR with the subfunction bit TF.AST.

The use of IO.ATA is enhanced by the addition of the TF.NOT and TF.XCC subfunction bits, which are described later in this section. You may include any or all the subfunctions described in this section with the IO.ATA function.

Unless the TF.XCC subfunction is specified, CTRL/C is trapped by the IO.ATA function task and does not reach the command line interpreter (CLI). Thus, any task that uses IO.ATA without the TF.XCC subfunction should recognize some input sequence as a request to terminate; otherwise, the CLI cannot be invoked to abort the task in case of difficulty.

Format

```
QIO$C  IO.ATA [ !TF.ESQ  ]
              [ !TF.NOT  ] ,lun,[efn],[pri],[isb], <[ast1],[parameter2],[ast2]>
              [ !TF.XCC  ]
```

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX-11D. Use a value of function 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast1

Specifies the entry point for an unsolicited input-character AST.

Either ast1 or ast2 is required.

Control passes to ast whenever an unsolicited character (other than CTRL/Q, CTRL/S, CTRL/X, or CTRL/O) is entered at the terminal. If ast2 is not specified, an unsolicited CTRL/C results in entering the AST specified in the ast parameter.

If TF.NOT is specified, after the AST has been effected, the AST becomes "disarmed" until a read request is issued by the task. If multiple characters are received before the read request is issued, they are stored in the type-ahead buffer. Once the read request is received, the contents of the type-ahead buffer, including the character causing the AST, is returned to the task; the AST is then "armed" again for new unsolicited input characters. If TF.NOT is not specified, every unsolicited character causes an AST.

Upon entry to the AST routines, the unsolicited character and parameter2 are in the top word on the stack, as shown in ast2. That word must be removed from the stack before exiting the AST.

parameter2

Indicates a value that you can specify to identify individual terminals in a multiterminal environment. Parameter2 is located in the high byte of SP+00.

ast2

Specifies the entry point for an unsolicited CTRL/C AST.

Either ast1 or ast2 is required.

If you specify the ast2 parameter, an unsolicited CTRL/C character results in entering the AST specified in that parameter. If ast2 is not specified, an unsolicited CTRL/C results in entering the AST specified in the ast parameter.

Upon entry to the AST routines, the unsolicited character and parameter2 are in the top word on the stack. That word must be removed from the stack before exiting the AST. The stack contents is as follows:

SP+10 Event flag mask word

SP+06 Program Section (PS) of task prior to AST

SP+04 Program Counter (PC) of task prior to AST

SP+02 Task's Directive Status Word (DSW)

SP+00 Unsolicited character in low byte

After the AST has been effected, the AST becomes "disarmed" until a read request is issued by the task. If multiple characters are received before the read request is issued, they are stored in the type-ahead buffer. Once the read request is received, the contents of the type-ahead buffer, including the character causing the AST, is returned to the task; the AST is then "armed" again for new unsolicited input characters. Thus, using the TF.NOT subfunction allows a task to monitor more than one terminal for unsolicited input without the need to read each terminal continuously for possible unsolicited input. Note that the TF.NOT subfunction cannot be used with the CTRL/C AST; an unsolicited CTRL/C character flushes the type-ahead buffer.

Either ast2 or TF.XCC can be used, but not both, in the same QIO$ request. If you specify both in the request, an IE.SPC error is returned.

See the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual* for further details on ASTs.

Subfunction Bits

TF.ESQ

Recognize Escape Sequences—This subfunction issued with IO.ATT or IO.ATA attaches a terminal and notifies the driver that it recognizes escape sequences entered at that terminal. Escape sequences are recognized only for solicited input (if a read was issued to the terminal). (See Section 2.7 for a discussion of escape sequences.)

If escape sequences are recognized, the sequence terminates input and a status code IS.ESC is returned. In addition, if uppercase-to-lowercase conversion is not enabled, the character ALTMODE (codes 175_8 or 176_8) is also treated as an escape character.

If the terminal has not been declared capable of generating escape sequences, IO.ATA!TF.ESQ has no effect other than attaching the terminal. No escape sequences are returned to the task because any ESCAPE sent by the terminal acts as a line terminator. The QIO$C SF.SMC function, the MCR command SET /ESCSEQ, or the DCL command SET/[NO]ESCAPE

declare the terminal capable of generating escape sequences (see Table 2–8 in Section 2.4.15, and see also Section 2.7).

TF.NOT

Notification of Unsolicited Input—Unsolicited input causes an AST and entry into the AST service routine in the task. When the full-duplex terminal driver receives unsolicited terminal input (except CTRL/C) and you used the TF.NOT subfunction with IO.ATA, the resulting AST serves only as notification of unsolicited terminal input; the terminal driver does not pass the character to the task. Upon entry to the AST service routine, the high byte of the first word on the stack identifies the terminal causing the AST (parameter2 in the IO.ATA function).

If TF.NOT is specified, after the AST has been affected, the AST becomes "disarmed" until a read request is issued by the task. If TF.NOT is not specified, every unsolicited character causes an AST.

Using the TF.NOT subfunction allows a task to monitor more than one terminal for unsolicited input without the need to read each terminal continuously for possible unsolicited input. Note that the TF.NOT subfunction cannot be used with the CTRL/C AST (ast2 in IO.ATA); an unsolicited CTRL/C character flushes the type-ahead buffer.

TF.XCC

Exclude CTRL/C from AST Notification—TF.XCC is for use with the IO.ATA function. When TF.XCC is included in the IO.ATA function, all characters (except CTRL/C) are handled in the manner previously described. CTRL/C marks the beginning of a command line interpreter (CLI) line that is processed by a CLI task, or, if CTRL/C abort is enabled, it aborts tasks active at the terminal. None of the characters of CLI input, including the CTRL/C, are sent to the task issuing the function.

Note that you can use either ast2 or TF.XCC, but not both, in the same QIO request. If both are specified in the request, an IE.SPC error is returned.

2.4.4 QIO$C IO.CCO—Cancel CTRL/O

The QIO$ IO.CCO macro directs the driver to write a logical block of data to the terminal regardless of a CTRL/O condition that may be in effect. If CTRL/O is in effect, it is canceled before the write occurs.

IO.CCO is equivalent to IO.WLB!TF.CCO.

Format

QIO$C IO.CCO [!TF.WAL / !TF.WBT] ,lun,[efn],[pri],[isb],[ast], <stadd,size,vfc>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast

Specify ast if you want to interrupt your task to execute special code upon completion of this I/O request. When the I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.

stadd

Specifies the starting address of the data buffer. Stadd may be on a byte boundary.

size

Specifies the size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

vfc

Specifies the cursor position.

If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter.

Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

Subfunction Bits

TF.WAL

Write All Characters—During the write-pass-all operation specified by this subfunction (as in IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation. It does not intercept control characters, and it does not keep track of cursor position. Long lines are not wrapped around if I/O wraparound has been selected.

TF.WBT

Breakthrough Write—This subfunction instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If another write function is currently in progress, it finishes the current request and the breakthrough write is the next write issued. Therefore, the TF.WBT subfunction cannot break through another breakthrough write that is in progress. The effect of this is that a CTRL/S can stop breakthrough write functions. Thus, it may be desirable for tasks to time out on breakthrough operations.

If a read is currently posted, the breakthrough write proceeds, and an automatic CTRL/R is performed to redisplay any input that write breakthrough received before the breakthrough write was effected (if the terminal is not in the full-duplex mode).

CTRL/O, if in effect, is canceled.

An escape sequence that was interrupted is rubbed out.

Privileged tasks may issue a breakthrough write to any terminal. In addition, a nonprivileged task may issue a breakthrough write to the task's terminal. (The privileged MCR command BRO (broadcast) uses IO.WBT.)

2.4.5 QIO$C IO.EIO—Extended I/O Functions

The QIO$C IO.EIO macro allows the use of additional I/O subfunctions. The design of the QIO$ macro, as used with the other QIO$ functions, allows a limited number of I/O subfunctions to be implemented. With IO.EIO, the address of an item list buffer (stadd) is contained in the macro statement. The item list buffer contains IO.EIO modifiers (recognizable as subfunctions), and it allows the use of a maximum of two words of I/O subfunction bits. See Figure 2–1, which shows the structure of the Item List 1 buffer for use with TF.RLB, and Figure 2–2, which shows the structure of the Item List 2 buffer for use with TF.WLB.

The QIO$C IO.EIO function reads from or writes to a terminal. The modifiers in the item list allow you to modify the nature or operation of that read or write. A read (TF.RLB) subfunction or write (TF.WLB) subfunction must be issued with the IO.EIO function. But both of these subfunctions cannot be executed together as a logical OR.

Note

IO.EIO function will not work if your terminal has been set as a remote terminal (RT) to another system. That is, after entering the following command and logging in to a remote terminal (RT), the terminal driver will reject a QIO by issuing an extended I/O request from the RT:

```
>SET HOST xxxxx
```

Formats

QIO$C IO.EIO!TF.RLB,lun,[efn],[pri],[isb],[ast], <stadd,size>

QIO$C IO.EIO!TF.WLB,lun,[efn],[pri],[isb],[ast], <stadd,size>

The TF.WLB and TF.RLB subfunctions each allow specific modifiers, which are located in the item list, to be used with them. They are listed as follows:

Subfunction	Modifiers			
TF.RLB	TF.BIN,	TF.RAL,	TF.RDI,	TF.RES,
	TF.RLU,	TF.RNE,	TF.RNF,	TF.RPR,
	TF.RPT,	TF.RST,[1]	TF.RTT,[1]	TF.TMO,
	TF.TNE,	TF.XOF		
TF.WLB	TF.CCO,	TF.RCU,	TF.WAL,	TF.WBT,
	TF.WIR			

[1] If both the TF.RST and TF.RTT modifiers are included, TF.RST supersedes the function of TF.RTT.

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX-11D. Use a function value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request by specifying ast. When the I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit asynchronous system trap (AST) processing.

stadd

Specifies the starting address of the item list of the length specified in size. The address of the item list must be word aligned and in the task's address space.

size

Specifies the size of the item list in bytes. The specified size for the IO.EIO!TF.RLB function cannot exceed 24_{10} bytes. The specified size for the IO.EIO!TF.WLB function cannot exceed 10_{10} bytes. The item list must be within the task's address space.

Subfunction Bits

TF.BIN

Binary Prompt (send prompt as pass all)—The prompt is sent to the terminal without interpretation by the driver. This is similar, for the prompt, to a write-pass-all operation.

TF.CCO

Cancel CTRL/O—The driver writes a logical block of data to the terminal regardless of a CTRL/O condition that may be in effect. If the CTRL/O is in effect, it is canceled before the write occurs.

TF.RAL

Read All Characters (Pass All)—This subfunction allows the passage of all characters to the requesting task. The driver does not intercept control characters. The characteristic TC.8BC, when set, allows the driver to pass 8 bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and are not interpreted by the driver.

TF.RCU

Restore Cursor Position—When defining cursor position in a function, you can use the TF.RCU subfunction to save the current cursor position. TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.

TF.RDI

Read with Default Input—The default input that you specified in the extended I/O item list is displayed as an input line at the start of the read on the terminal. You may change this line or use it as input to the system. This subfunction is for use with the extended I/O function (IO.EIO) only.

TF.RES

Read with Escape Sequence Processing Enabled—This subfunction enables escape sequence recognition for the read operation in extended I/O; it is effective for one read only.

TF.RLU

Read with Conversion From Lowercase To Uppercase—The task that uses this subfunction gets input in the buffer in uppercase. This subfunction is used with the extended I/O (IO.EIO) function only.

TF.RNE

Read with No Echo—This subfunction reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information. CTRL/R is ignored while Read With No Echo is in progress.

TF.RNF

Read with No Filter—This subfunction reads and passes through CTRL/U, CTRL/R, and DELETE characters as normal characters. It is for use with the extended I/O (IO.EIO) function only.

TF.RPR

Read After Prompt—This subfunction is for use with the extended I/O (IO.EIO) function only. The TF.RPR subfunction causes a prompt to be sent to the terminal and immediately follows it with a read function at the terminal. The TF.RPR acts as an IO.WLB followed by IO.RLB. However, TF.RPR differs from the combination of those two functions as follows:

- System overhead is lower with the TF.RPR because only one QIO$ TF.RPR is processed.
- When using the TF.RPR function, there is no "window" during which a response to the prompt may be ignored. Such a window occurs if the task uses IO.WLB followed by an IO.RLB, because no read may be posted at the time the response is received.
- If the issuing task is checkpointable, it can be checkpointed during both the prompt and the read requested by the TF.RPR.
- A CTRL/O that may be in effect prior to issuing the TF.RPR is canceled before the prompt is written.

Note

If a TF.RPR function is in progress when the driver receives a CTRL/R or CTRL/U, the prompt is redisplayed.

TF.RPT

Read in Pass-Through Mode—This subfunction passes all characters except XON/XOFF. It allows the passage of all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass 8 bits instead of 7. The driver intercepts the control characters CTRL/S and CTRL/Q. Other control characters, for example, CTRL/C, CTRL/O, and CTRL/Z, are passed to the task and are not interpreted by the driver. This subfunction modifier is for use with the IO.EIO!TF.RLB function only.

TF.RST

Read with Special Terminators—Special characters in the ranges 0 to 037_{10} and 175_{10} to 177_{10} terminate the read. The driver does not interpret the terminating character. For example, a DELETE or RUBOUT 177_{10} does not erase, and a CTRL/C does not produce a CLI prompt or abort tasks active at the terminal if CTRL/C abort is enabled. CTRL/U and CTRL/R do not perform their usual functions either. All control characters are terminators.

TF.RST sets TF.TNE by default, which means that terminators are not echoed on the terminal screen.

If uppercase-to-lowercase conversion is disabled, characters 175_{10} and 176_{10} do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017_{10}, 021_{10}, and 023_{10}, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner.

Exercise great care when using IO.RAL and TF.RST together. Obscure problems can result if you use them in this way.

TF.RTT

Read with Specified Terminator Table—This subfunction is for use with the IO.EIO extended I/O function only. Control characters function normally with the TF.RTT subfunction. Terminators echo by default. The additional use of subfunction TF.TNE prevents the echoing of terminators on the terminal screen. If you want to use special control characters

as terminators, their normal function should be disabled with the TF.RNF subfunction or the TC.PTH characteristic. The terminator table (a bit mask table) length can be from 1_{10} to 32_{10} bytes where bit 0 is a null character, bit 1 is a CTRL/A, and so forth. The terminator table address is in the item list of the IO.EIO function. To use ASCII characters 128_{10} to 255_{10}, the characteristic TC.8BC must be set.

TF.TMO

Read with Timeout—This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time.

Specify the timeout count in seconds. Timeout is the maximum time allowed between two input characters before the read is aborted. The maximum timeout value is 255_{10} intervals.

If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer.

If you need more than 255_{10} seconds, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.

TF.TNE

Read Terminators with No Echo—This subfunction allows reading terminator characters from the terminal without their being echoed on the terminal screen as they are entered. It is for use with the extended I/O function IO.EIO only.

TF.WAL

Write All Characters—During the write-pass-all operation specified by this subfunction (as in IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation. It does not intercept control characters, and it does not keep track of cursor position. Long lines are not wrapped around if I/O wraparound has been selected.

TF.WBT

Breakthrough Write—Instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If another write function is currently in progress, it finishes the current request and the breakthrough write is the next write issued. Therefore, the TF.WBT subfunction cannot break through another breakthrough write that is in progress. The effect of this is that a CTRL/S can stop breakthrough write functions. Thus, it may be desirable for tasks to time out on breakthrough operations.

If a read is currently posted, the breakthrough write proceeds, and an automatic CTRL/R is performed to redisplay any input that was received before the breakthrough write was effected (if the terminal is not in the full-duplex mode).

CTRL/O, if in effect, is canceled.

An escape sequence that was interrupted is deleted.

Privileged tasks may issue a breakthrough write to any terminal. In addition, a nonprivileged task may issue a breakthrough write to the task's terminal.

TF.WIR

Write with Input Redisplayed—This subfunction performs a write to the terminal. If a read is in progress at the terminal and you have entered characters in the input line, the prompt and the characters are redisplayed at the end of the write.

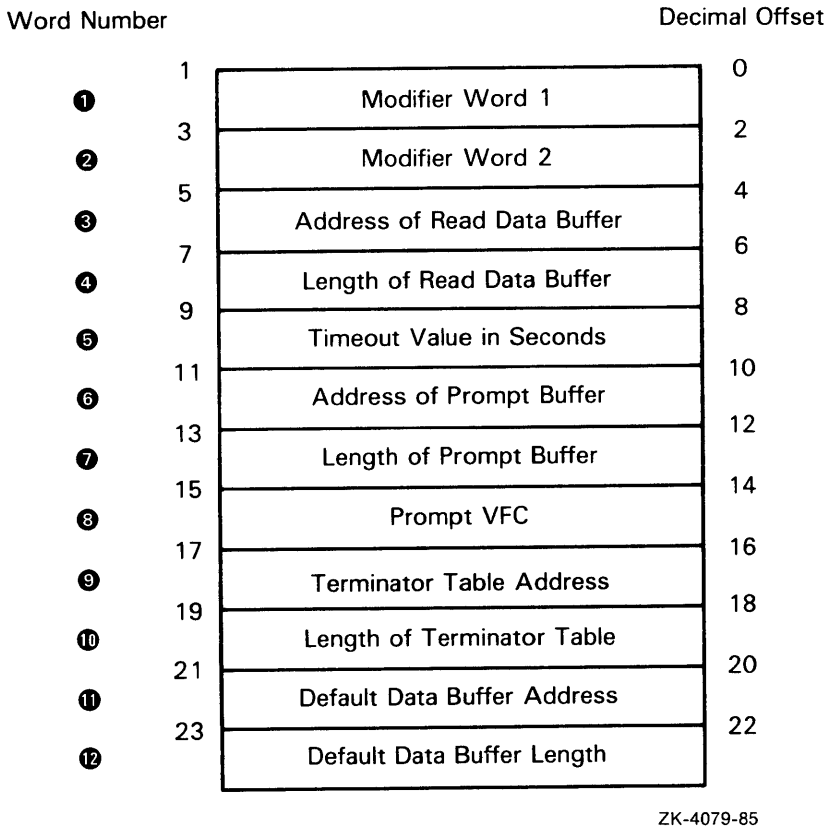
TF.XOF

Send XOFF—This subfunction causes the driver to send an XOFF to the terminal after it is read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

2.4.5.1 Item List 1 for IO.EIO!TF.RLB

Figure 2–1 shows the structure of the Item List 1 buffer. You should use the Item List 1 buffer when you use the TF.RLB function with IO.EIO. Modifier word 2 is currently not used but must be 0. All the other fields in the item list must be present if they fall before the last pertinent field. Thus, if a read with prompt (TF.RPR) is not being performed, words 6, 7, and 8 are not used.

Figure 2–1: Structure of the Item List 1 Buffer



❶ Modifiers (subfunctions) of the group of additional modifiers allowed for any I/O read function.

❷ Currently must be 0.

❸ The starting address of the read data buffer. The read data buffer may be on a byte boundary.

❹ The size of the read data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

❺ For use with TF.TMO. TF.TMO must be in modifier word 1.

❻ For use with TF.RPR and contains the starting address of the prompt buffer. TF.RPR must be in modifier word 1. The prompt buffer may be on a byte boundary.

❼ For use with TF.RPR. The size of the prompt buffer in bytes. The buffer must be within the task's address space. The specified size must be greater than 0 and less than or equal to 8128 bytes.

❽ For use with TF.RPR. The vfc parameter normally specifies cursor position.

If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter.

Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

❾ For use with TF.RTT. The TF.RTT function must be in modifier word 1. The table (1_{10} to 32_{10} bytes) starts at the address specified by the table address. The first word contains bits that represent the first 16 ASCII character codes (0 to 17_{10}); similarly, the second word contains bits that represent the next 16 character codes (20_{10} to 37_{10}), and so forth, through the sixteenth word, bit 15, which represents character code 377_{10}. For example, to specify the percent sign (%) symbol (code 045_{10}) as a read terminator character, set bit 05 in the third word, because the third word of the table contains bits representing character codes 40_{10} to 57_{10}.

The terminal must be set for read-pass-all operation (TC.BIN=1) or to read-pass 8 bits (TC.8BC) if you want to use any of the following characters as terminator characters:

- CTRL/S (023)
- CTRL/Q (021)
- Any characters whose codes are greater than 177_{10}

❿ Length of the terminator table specified in the terminator table address field.

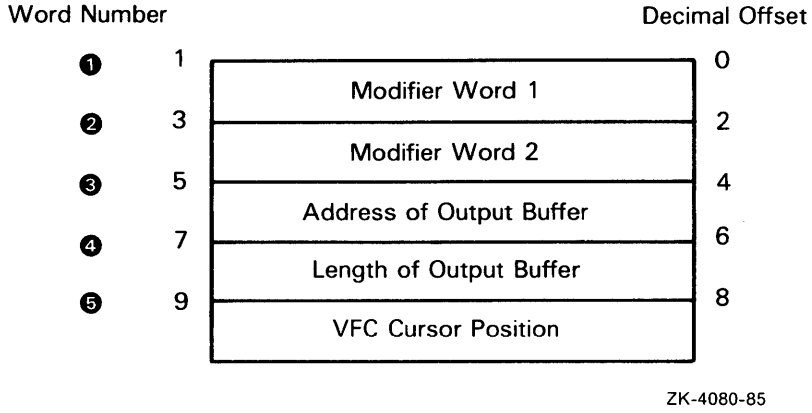
⓫ For use with TF.RDI. TF.RDI must be in modifier word 1. This buffer contains the default input that is to be displayed on the terminal.

⓬ For use with TF.RDI. This word contains the length of the buffer at the address specified in the default data buffer address field.

2.4.5.2 Item List 2 for IO.EIO!TF.WLB

You should use the Item List 2 buffer when you use the TF.WLB function with IO.EIO. Modifier word 2 is not used currently but must be 0. All the other fields in the item list must be present except the VFC cursor position, which is optional. Item list 2 is shown in Figure 2–2.

Figure 2–2: Structure of the Item List 2 Buffer



❶ Modifiers (subfunctions) of the group of modifiers allowed for I/O write functions.

❷ Currently must be 0.

❸ The starting address of the write data buffer. The address may be on a byte boundary.

❹ The size of the stadd buffer in an even number of bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

❺ The vfc parameter normally specifies cursor position.

If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter.

Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

2.4.6 QIO$C IO.GTS—Get Terminal Support

The QIO$C IO.GTS macro returns information to a 4-word buffer that specifies which system generation options are part of the terminal driver. Only two of these words are currently defined. Table 2–7 gives details for these words. The IO.GTS function is a system generation option. If IO.GTS is issued on a system without IO.GTS support, IE.IFC is returned in the I/O status block (IOSB).

Format

QIO$C IO.GTS,lun,[efn],[pri],[isb],[ast], <stadd,size>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX–11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request by specifying ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit asynchronous system trap (AST) processing.

stadd

Specifies the starting address of the data buffer. The address must be word aligned.

size

Specifies the size of the stadd data buffer in bytes. The specified size must be 4 bytes. The buffer must be within the task's address space.

The various symbols used by the IO.GTS, SF.GMC, and SF.SMC functions are defined in a system module, TTSYM. These symbols include F1.xxx and F2.xxx (Table 2–7); TC.xxx (Table 2–8); T.xxxx (Table 2–9); and the SE.xxx status returns described in Table 2–10. These symbols may be defined locally within a code module by using the following:

```
.MCALL   TTSYM$
  .
  .
  .
TTSYM$
```

Symbols that are not defined locally are automatically defined by the Task Builder (TKB).

Octal values shown for the symbols are subject to change. Therefore, only the symbolic names should be used.

Table 2–7: Information Returned by Get Terminal Support (IO.GTS) QIO$

Bit	Octal Value	Mnemonic	Meaning When Set to 1
Word 0 of Buffer:			
0	1	F1.ACR	Automatic CR/LF on long lines
1	2	F1.BTW	Breakthrough write
2	4	F1.BUF	Checkpointing during terminal input
3	10	F1.UIA	Unsolicited input-character AST
4	20	F1.CCO	Cancel CTRL/O before writing
5	40	F1.ESQ	Recognize escape sequences in solicited input
6	100	F1.HLD	Hold-screen mode
7	200	F1.LWC	Lowercase-to-uppercase conversion
8	400	F1.RNE	Read with no echo
9	1000	F1.RPR	Read after prompting
10	2000	F1.RST	Read with special terminators
11	4000	F1.RUB	CRT rubout
12	10000	F1.SYN	CTRL/R terminal synchronization
13	20000	F1.TRW	Read all and write all
14	40000	F1.UTB	Input characters buffered in task's address space
15	100000	F1.VBF	Variable-length terminal buffers

Table 2–7 (Cont.): Information Returned by Get Terminal Support (IO.GTS) QIO$

Bit	Octal Value	Mnemonic	Meaning When Set to 1
Word 1 of Buffer:			
0	1	F2.SCH	Set characteristics QIO$ (SF.SMC)
1	2	F2.GCH	Get characteristics QIO$ (SF.GMC)
2	4	F2.DCH	Dump/restore characteristics
3	10	F2.DKL	Historical RSX–11D or IAS IO.KIL
4	20	F2.ALT	ALTMODE is echoed
5	40	F2.SFF	Form feed can be simulated
6	100	F2.CUP	Cursor positioning
7	200	F2.FDX	Full-duplex terminal driver
8	400	F2.EIO	Extended I/O
9	1000	F2.NCT	Network command terminal support

2.4.7 QIO$C IO.HNG—Disconnect a Terminal

The QIO$C IO.HNG macro disconnects a terminal that is on a remote line or on a DECnet link.

A nonprivileged task can issue an IO.HNG request for its own terminal (TI) only. A privileged task can issue IO.HNG to any terminal.

Format

```
QIO$C   IO.HNG,lun,[efn],[pri],[isb],[ast]
```

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX–11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request specifying ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.

2.4.8 QIO$C IO.RAL—Read All Characters Without Interpretation

The QIO$C IO.RAL macro causes the driver to pass all characters that were read to the requesting task. The driver does not intercept control characters. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the program and are not interpreted by the driver.

Note

IO.RAL echoes the characters that are read. To read all characters without echoing, use IO.RAL!TF.RNE.

IO.RAL is equivalent to IO.RLB used in a logical OR with the subfunction bit TF.RAL. The IO.RAL function can be terminated only by a full character count (input buffer full).

Format

$$\text{QIO\$C}\quad \text{IO.RAL}\left[\begin{array}{l}\text{!TF.RNE}\\ \text{!TF.RST}\\ \text{!TF.TMO}\\ \text{!TF.XOF}\end{array}\right], \text{lun,[efn],[pri],[isb],[ast], <stadd,size,[tmo]>}$$

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request by specifying ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit asynchronous system trap (AST) processing.

stadd

Specifies the starting address of the data buffer. Stadd may be on a byte boundary.

size

Specifies the size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

tmo

Specifies the optional timeout count for use with the TF.TMO subfunction.

Subfunction Bits

TF.RNE

Read with No Echo—This subfunction reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information. CTRL/R is ignored while Read with No Echo (TF.RNE) is in progress.

TF.RST

Read with Special Terminators—Special characters in the ranges 0 to 037 and 175 to 177 terminate the read. The driver does not interpret the terminating character. For example, a DELETE (or RUBOUT) does not erase, and a CTRL/C does not produce a CLI prompt or abort tasks active at the terminal if CTRL/C abort is enabled. Also CTRL/U and CTRL/R do not perform their usual functions either. All control characters are terminators.

If uppercase-to-lowercase conversion is disabled, characters 175 and 176 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner.

Exercise great care when using IO.RAL and TF.RST together. Obscure problems can result if you use them in this way.

TF.TMO

Read with Timeout—This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time.

Specify the timeout count in 10-second intervals. Timeout is the maximum time allowed between two input characters before the read is aborted. The maximum timeout value is 255_{10} intervals.

If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer.

If you need more than 255_{10} intervals, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.

TF.XOF

Send XOFF—The driver sends an XOFF to the terminal after its read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

2.4.9 QIO$C IO.RNE—Read Input Without Echoing

The IO.RNE function reads terminal input characters without echoing the characters back to the terminal for display. You can use this feature when typing sensitive information (for example, a password or combination) or when reading a badge with the RT02-C terminal.

(Note that the no-echo mode can also be selected with the SF.SMC function; see Table 2–8 in Section 2.4.15, bit TC.NEC.)

CTRL/R is ignored while an IO.RNE is in progress.

The IO.RNE function is equivalent to IO.RLB in a logical OR with the subfunction bit TF.RNE.

Format

QIO$C IO.RNE [!TF.RAL / !TF.RST / !TF.TMO / !TF.XOF] ,lun,[efn],[pri],[isb],[ast], <stadd,size,[tmo]>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX–11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request by specifying ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit asynchronous system trap (AST) processing.

stadd

Specifies the starting address of the data buffer. Stadd may be on a byte boundary.

size

Specifies the size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

tmo

Specifies the optional timeout count for use with the TF.TMO subfunction.

Subfunction Bits

TF.RAL

Read All Characters (Pass All)—This subfunction allows the driver to pass all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass 8 bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and are not interpreted by the driver.

Exercise great care when using TF.RAL (read all) and TF.RST (read with special terminators) together. Obscure problems can result if you use them in this way.

TF.RST

Read with Special Terminators—Special characters in the ranges 0 to 037 and 175 to 177 terminate the read. The driver does not interpret the terminating character. For example, a DELETE (or RUBOUT) does not erase, and a CTRL/C does not produce a command line interpreter (CLI) prompt or abort tasks active at the terminal if CTRL/C abort is enabled. Also CTRL/U and CTRL/R do not perform their usual functions either. All control characters are terminators.

If uppercase-to-lowercase conversion is disabled, characters 175 and 176 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner.

Exercise great care when using TF.RAL (read all) and TF.RST (read with special terminators) together. Obscure problems can result if you use them in this way.

TF.TMO

Read with Timeout—This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time.

Specify the timeout count in 10-second intervals. Timeout is the maximum time allowed between two input characters before the read is aborted. The maximum timeout value is 255_{10} intervals.

If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer.

If you need more than 255_{10} intervals, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.

TF.XOF

Send XOFF—The driver sends an XOFF to the terminal after it is read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

2.4.10 QIO$C IO.RPR—Read with Prompt

The QIO$C IO.RPR macro sends a prompt to the terminal and immediately follows it with a read function at the terminal. The IO.RPR functions as an IO.WLB (write a prompt to the terminal) followed by IO.RLB. However, IO.RPR differs from the combination of those two functions as follows:

- System overhead is lower with the IO.RPR because only one QIO$ is processed.
- When you use the IO.RPR function, there is no "window" during which a response to the prompt may be ignored. Such a window occurs if the task uses IO.WLB/IO.RLB, because no read may be posted at the time the response is received.
- If the issuing task is checkpointable, it can be checkpointed during both the prompt and the read requested during the IO.RPR.
- A CTRL/O that may be in effect prior to issuing the IO.RPR is canceled before the prompt is written.

Subfunction bits may be excuted as a logical OR with IO.RPR to write the prompt as a "write all" (TF.BIN) and to send XOFF after the read (TF.XOF). In addition, your task can use TF.RAL, TF.RNE, and TF.RST with IO.RPR.

Note

If an IO.RPR function is in progress when the driver receives a CTRL/R or CTRL/U, the prompt is redisplayed.

Format

QIO$C IO.RPR [!TF.BIN | !TF.RAL | !TF.RNE | !TF.RST | !TF.TMO | !TF.XOF] ,lun,[efn],[pri],[isb],[ast], <stadd,size,[tmo],pradd,prsize,vfc>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request by specifying ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.

stadd

Specifies the starting address of the data buffer. Stadd may be on a byte boundary.

size

Specifies the size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

tmo

Specifies the optional timeout count for use with the TF.TMO subfunction.

pradd

Specifies the starting address of the byte buffer where the prompt is stored.

prsize

Specifies the size of the pradd prompt buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

vfc

Specifies the cursor position.

If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter.

Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

Subfunction Bits

TF.BIN

Binary Prompt (send prompt as pass all)—As used in IO.RPR, the TF.BIN subfunction results in a "binary" prompt; that is, a prompt is sent to the terminal by the driver with no character interpretation (as if it were issued as an IO.WAL). The read follows the binary prompt.

TF.RAL

Read All Characters (Pass All)—The driver passes all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass 8 bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and are not interpreted by the driver.

Exercise great care when using TF.RAL (read all) and TF.RST (read with special terminators) together. Obscure problems can result if you use them in this way.

TF.RNE

Read with No Echo—This subfunction reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information. CTRL/R is ignored while TF.RNE is in progress.

TF.RST

Read with Special Terminators—Special characters in the ranges 0 to 037 and 175 to 177 terminate the read. The driver does not interpret the terminating character. For example, a DELETE (or RUBOUT) does not erase, and a CTRL/C does not produce a CLI prompt or abort tasks active at the terminal if CTRL/C abort is enabled. Also CTRL/U and CTRL/R do not perform their usual functions either. All control characters are terminators.

TF.RST sets TF.TNE by default, which means that terminators are not echoed on the terminal screen.

If lowercase-to-uppercase conversion is disabled, characters 175 and 176 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner.

Exercise great care when using TF.RAL and TF.RST together. Obscure problems can result if you use them in this way.

TF.TMO

Read with Timeout—This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time.

Specify the timeout count in 10-second intervals. Timeout is the maximum time allowed between two input characters before the read is aborted. The maximum timeout value is 255_{10} intervals.

If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer.

If you need more than 255_{10} intervals, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.

TF.XOF

Send XOFF—The driver sends an XOFF to the terminal after its prompt-and-read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

2.4.11 QIO$C IO.RST—Read Logical Block with Special Terminators

A QIO$C IO.RST reads a block of data from the terminal. This function is equivalent to an IO.RLB!TF.RST. Certain special characters in the ranges 0 to 037 and 175 to 177 terminate the read. The driver does not interpret the terminating character. For example, a DELETE or RUBOUT 177_{10} does not erase, and a CTRL/C does not produce a command line interpreter (CLI) prompt or abort tasks active at the terminal if CTRL/C abort is enabled. Also CTRL/U and CTRL/R do not perform their usual functions. All control characters are terminators.

If lowercase-to-uppercase conversion is disabled, characters 175 and 176 do not act as terminators. CTRL/O, CTRL/Q, and CTRL/S (017, 021, and 023, respectively) are not special terminators. The driver interprets them as output control characters in a normal manner.

Upon successful completion of an IO.RST request that was not terminated by filling the input buffer, the first word of the I/O status block (IOSB) contains the terminating character in the high byte and the IS.SUC status code in the low byte. The second word contains the number of bytes contained in a buffer. The terminating character is not put in the buffer.

Format

QIO$C IO.RST [!TF.RAL | !TF.RNE | !TF.TMO | !TF.XOF] ,lun,efn,[pri],[isb],[ast], <stadd,size,tmo>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX–11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request by specifying ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit asynchronous system trap (AST) processing.

stadd

Specifies the starting address of the data buffer. Stadd may be on a byte boundary.

size

Specifies the size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

tmo

Specifies the optional timeout count for use with the TF.TMO subfunction.

Subfunction Bits

TF.RAL

Read All Characters (Pass All)—The driver passes all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass 8 bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and are not interpreted by the driver.

Exercise great care when using TF.RAL (read all) and TF.RST (read with special terminators) together. Obscure problems can result if you use them in this way.

TF.RNE

Read with No Echo—This subfunction reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information.

TF.TMO

Read with Timeout—This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time.

Specify the timeout count in 10-second intervals. Timeout is the maximum time allowed between two input characters before the read is aborted. The maximum timeout value is 255_{10} intervals.

If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer.

If you need more than 255_{10} intervals, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.

TF.XOF

Send XOFF—The driver sends an XOFF to the terminal after it is read. The XOFF (CTRL/S) may have the effect of inhibiting input from the terminal, if the terminal recognizes XOFF for this purpose. TF.XOF is ignored when full-duplex I/O is in use.

2.4.12 QIO$ IO.RTT—Read with Terminator Table

The QIO$C IO.RTT macro reads characters like the QIO$C IO.RLB macro, except that a character that you have previously specified terminates the read operation. The specified character's code can range from 0 to 377_8. You can specify it by setting a bit in a 16-word table, which you specify, that corresponds to the desired character. Multiple characters can be specified by setting their corresponding value.

The 16-word table starts at the address specified by the table parameter. The first word contains bits that represent the first 16 ASCII character codes (0 to 17); similarly, the second word contains bits that represent the next 16 character codes (20 to 37), and so forth, through the sixteenth word, bit 15, which represents character code 377. For example, to specify the percent (%) symbol (code 045) as a read terminator character, set bit 05 in the third word, because the third word of the table contains bits representing character codes 40 to 57.

If you want to use the CTRL/S (023), CTRL/Q (021), or any characters greater than 177 as the terminator characters, the terminal must be set to allow a read-pass-all operation (TC.BIN=1), or read-pass 8 bits (TC.8BC), as listed in Table 2–8 in Section 2.4.15.

The optional timeout count parameter may be included as desired.

Format

QIO$C IO.RTT [!TF.RAL / !TF.RCU / !TF.RNE / !TF.TMO] ,lun,[efn],[pri],[isb],[ast], <stadd,size,[tmo],table>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX–11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request by specifying ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit asynchronous system trap (AST) processing.

stadd

Specifies the starting address of the data buffer. Stadd may be on a byte boundary.

size

Specifies the size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

tmo

Specifies the optional timeout count for use with the TF.TMO subfunction.

table

Specifies the address of the 16-word, user-specified terminator table that you create in your task.

Subfunction Bits

TF.RAL

Read All Characters (Pass All)—The driver passes all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass 8 bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and are not interpreted by the driver.

Exercise great care when using TF.RAL (read all) and TF.RST (read with special terminators) together. Obscure problems can result if you use them in this way.

TF.RCU

Restore Cursor Position—When defining cursor position in a function, you can use the TF.RCU subfunction to save the current cursor position. TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.

TF.RNE

Read with No Echo—This subfunction reads terminal input characters without echoing the characters back to the terminal for immediate display. You can use this feature when typing sensitive information. CTRL/R is ignored while TF.RNE is in progress.

TF.TMO

Read with Timeout—This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time.

Specify the timeout count in 10-second intervals. Timeout is the maximum time allowed between two input characters before the read is aborted. The maximum timeout value is 255_{10} intervals.

If 0 is specified, the read times out immediately after reading any data that may be in the type-ahead buffer. In other words, if you enter a 0, no time is allowed for you to enter characters, and all characters are read from the type-ahead buffer.

If you need more than 255_{10} intervals, issue an asynchronous QIO$ request followed by a Mark Time directive (MRKT$) for the required interval. Specify different event flags in the two directives and, after issuing them, wait for the logical OR of the two event flags.

2.4.13 QIO$C IO.WAL—Write a Logical Block and Pass All Characters

The QIO$C IO.WAL macro causes the driver to pass all output from the buffer without interpretation. It does not intercept control characters. Long lines are not wrapped around if I/O wraparound has been selected.

IO.WAL is equivalent to the IO.WLB!TF.WAL function.

Format

QIO$C IO.WAL { !TF.CCO / !TF.RCU / !TF.WBT } ,lun,[efn],[pri],[isb],[ast], <stadd,size,vfc>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request by specifying ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.

stadd

Specifies the starting address of the data buffer. Stadd may be on a byte boundary.

size

Specifies the size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

vfc

Specifies the cursor position.

If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter.

Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

Subfunction Bits

TF.CCO

Cancel CTRL/O—The driver writes a logical block of data to the terminal regardless of a CTRL/O condition that may be in effect. The CTRL/O, if in effect, is canceled before the write occurs.

During a write-pass-all operation (IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation; it does not keep track of cursor position.

TF.RCU

Restore Cursor Position—When defining cursor position in a function, you can use the TF.RCU subfunction to save the current cursor position. TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.

During a write-pass-all operation (IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation; it does not keep track of cursor position.

TF.WBT

Breakthrough Write—This subfunction instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If another write function is currently in progress, it finishes the current request and the breakthrough write is the next write issued. Therefore, the TF.WBT subfunction cannot break through another breakthrough write that is in progress. The effect of this is that a CTRL/S can stop breakthrough write functions. Thus, it may be desirable for tasks to time out on breakthrough write operations.

If a read is currently posted, the breakthrough write proceeds, and an automatic CTRL/R is performed to redisplay any input that was received before the breakthrough write was effected (if the terminal is not in the full-duplex mode).

CTRL/O, if in effect, is canceled.

An escape sequence that was interrupted is deleted.

Privileged tasks may issue a breakthrough write to any terminal. In addition, a nonprivileged task may issue a breakthrough write to the task's terminal.

During a write-pass-all operation, (IO.WAL or IO.WLB!TF.WAL) the terminal driver outputs characters without interpretation; it does not keep track of cursor position.

2.4.14 QIO$C IO.WBT—Break Through to Write a Logical Block

The QIO$C IO.WBT macro instructs the driver to write the buffer regardless of the I/O status of the receiving terminal. If an IO.WBT function is issued on a system that does not support IO.WBT, it is treated as an IO.WLB function. The IO.WBT macro works as follows:

- If another write function is currently in progress, it finishes the current request and the IO.WBT is the next write issued. The effect of this is that a CTRL/S can stop IO.WBT functions. Therefore, it may be desirable for tasks to time out on IO.WBT operations.
- If a read is currently posted, the IO.WBT proceeds, and an automatic CTRL/R is performed to redisplay any input that was received before the breakthrough write was effected (if the terminal is not in the full-duplex mode).
- If CTRL/O is in effect, it is canceled.
- An escape sequence that was interrupted is deleted.

An IO.WBT function cannot break through another IO.WBT that is in progress.

Privileged tasks may issue a breakthrough write to any terminal. In addition, a nonprivileged task may issue a breakthrough write to the task's terminal. The privileged MCR command BRO (broadcast) uses IO.WBT.

Format

QIO$C IO.WBT [!TF.CCO | !TF.RCU | !TF.WAL] ,lun,[efn],[pri],[isb],[ast], <stadd,size,vfc>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX–11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request by specifying ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit asynchronous system trap (AST) processing.

stadd

Specifies the starting address of the data buffer. Stadd may be on a byte boundary.

size

Specifies the size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space.

vfc

Specifies the cursor position.

If the parameter defines cursor position, the high byte must be a nonzero number. The low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1,1). The driver outputs cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

However, the parameter is interpreted as a vertical forms control (vfc) parameter if its high byte is 0. See Section 2.8 for more information about the characters your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter.

Terminal-independent cursor control capability is provided at system generation time. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use.

Subfunction Bits

TF.CCO

Cancel CTRL/O—The driver writes a logical block of data to the terminal regardless of a CTRL/O condition that may be in effect. If the CTRL/O is in effect, it is canceled before the write occurs. The IO.WBT function implies the subfunction TF.CCO; therefore, using IO.WBT!TF.CCO is redundant.

TF.RCU

Restore Cursor Position—When defining cursor position in a function, you can use the TF.RCU subfunction to save the current cursor position. TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.

TF.WAL

Write All Characters—During a write-pass-all operation (as in IO.WAL or IO.WLB!TF.WAL), the terminal driver outputs characters without interpretation. It does not intercept control characters, and it does not keep track of cursor position. Long lines are not wrapped around if I/O wraparound has been selected.

2.4.15 QIO$C SF.GMC—Get Multiple Characteristics

The QIO$ SF.GMC macro returns terminal information into a specified buffer. Table 2–8 shows the terminal characteristics that can be obtained with the QIO$ SF.GMC macro and set with the QIO$ SF.SMC macro.

Format

QIO$C SF.GMC,lun,[efn],[pri],[isb],[ast], <stadd,size>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX–11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1, but consider the following exception.

For SF.GMC, the contents of the I/O Status Block (IOSB) is different from that described in Chapter 1. The first word of the status block is the same as the first word of the status block described in Chapter 1. However, the second word is not the same. For SF.GMC or SF.SMC, the second word contains the number of bytes in the specified user buffer that were sucessfully processed. For example, if you have a characteristic in the buffer that caused an error, ISB+2 (the second word) will contain the offset to the characteristic.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request by specifying ast. When the I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit AST processing.

stadd

Specifies the starting address of a data buffer of length "size" bytes. For most characteristics, each word in the buffer has the following form:

```
.BYTE characteristic-name
.BYTE 0
```

characteristic-name

Specifies one of the bit names that is given in Table 2–8. The value returned in the high byte of each byte-pair is 1 if the characteristic is true for the terminal and the value is 0 if the characteristic is not true. Some characteristics require a different buffer format. These formats are described in Section 2.4.16.1.

size

Specifies the size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space. For SF.GMC, size must be an even value. For the TC.TTP characteristic (terminal type), one of the values shown in Table 2–9 is returned in the high byte.

Table 2–8: Terminal Characteristics for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (If Asserted)	MCR Command
TC.ABD	77	Specifies autobaud detection.	SET /ABAUD=TTnn:
TC.ACD	103	Specifies the ancillary control driver. The value is determined by system manager.	–
TC.ACR	24	Specifies wraparound mode.	SET /WRAP=TTnn:
TC.ANI	122	Specifies an ANSI CRT terminal.	SET /ANSI=TTnn:
TC.ASP	76	Specifies remote line answer speed and initial speed over dial-up line.	SET /REMOTE=TTnn:speed
TC.AVO	123	Specifies VT100-family terminal display.	SET /AVO=TTnn:
TC.BIN	65	Specifies binary input mode (read-pass-all). No characters are interpreted as control characters.	SET /RPA=TTnn:
TC.BLK	42	Specifies that the terminal is capable of block mode transfers.	SET /BLKMOD=TTnn:
TC.CLN	152	Specifies 7- or 8-bit character size at the hardware level.	SET /CHAR_LENGTH
TC.CTS	72	Suspends output to a terminal. The task can cancel an input ^S or get the current state of the terminal with regard to ^S or ^Q, as follows: 0 = Resume 1 = Suspend	–
TC.DEC	124	Specifies a DIGITAL CRT terminal.	SET /DEC=TTnn:

Table 2–8 (Cont.): Terminal Characteristics for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (If Asserted)	MCR Command
TC.DLU[1]	41	Specifies a dial-up line as follows: 0 = Local line 1 = Remote line 2 = Remote line with autocall enabled	SET /REMOTE=TTnn:
TC.EDT	125	Specifies terminal performs editing functions.	SET /EDIT=TTnn:
TC.EPA	40	Specifies the following when TC.PAR is enabled: 0 = Odd parity 1 = Even parity	–
TC.ESQ	35	Specifies input escape sequence recognition.	SET /ESCSEQ=TTnn:
TC.FDX	64	Specifies full-duplex mode.	SET /FDX=TTnn:
TC.HFF	17	Specifies hardware form-feed capability. (If 0, form feeds are simulated using TC.LPP.)	SET /FORMFEED=TTnn:
TC.HFL	13	Specifies the number of fill characters to insert after a carriage return (0–7=x). (Use a value of 7 for the LA30-S.)	SET /HFILL=TTnn:x
TC.HHT	21	Specifies horizontal tab capability. (If 0, horizontal tabs are simulated using spaces.)	SET /HHT=TTnn:
TC.HLD	44	Specifies hold screen mode and indicates the terminal has the ability to hold screen. This feature is not supported over the network (NCT).	SET /HOLD=TTnn:
TC.HSY	132	Specifies host-to-terminal synchronization. This bit is sent when resources are low. XON is sent when resources are high. XOFF prevents terminal character input, as follows: 0 = No flow control 1 = Flow control exerted	XOFFSET /HSYNC=TTnn:
TC.ICS	141	Indicates change in type-ahead buffer (input count state).	

[1] A program can enable the autocall feature of the DF03 modem by setting TC.DLU to a value of 2. Autocall allows you to use the terminal to dial out of the computer. (This is in addition to receiving incoming calls.) While in this mode, read and write requests are serviced even when a line is not in use. Consequently, I/O requests do not fail when the line is hung-up, which is the case for remote lines (TC.DLU=1).

Table 2–8 (Cont.): Terminal Characteristics for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (If Asserted)	MCR Command
TC.ISL	6	Gets MUX subline (=0–15) on interface to which user is connected (SF.GMC only).	–
TC.LPP	2	Specifies page length ($1–255_{10}$=x).	SET /LINES=TTnn:x
TC.MAP	154	Specifies LAT application terminal. Broadcasting is disabled and no unsolicited input is accepted by default.	
TC.MHU	145	Declares modem hang-up asynchronous system trap (AST). Specifies address of AST activated by lost carrier.	–
TC.NBR	102	Indicates broadcast disabled.	SET /NOBRO=TTnn:
TC.NEC	47	Indicates echo suppressed.	SET /NOECHO=TTnn:
TC.OOB	140	Specifies out-of-band characters and whether they are included in the type-ahead buffer, and whether they are to clear the type-ahead buffer.	–
TC.PAR	37	Generates and checks parity.	SET /PARITY=TTnn:
TC.PPT	147	Specifies that the terminal has a printer port.	SET /PRINTERPORT=TTnn:
TC.PRI	51	Specifies that the terminal is privileged (SF.GMC only).	SET /PRIV=TTnn:
TC.PTH	146	Specifies pass through enable. Only CTRL/S and CTRL/Q are honored as follows: 1 = Pass Through 0 = Default; no pass through	SET /PASTHRU=TTnn:
TC.QDP	153	Specifies queue-depth of LAT application terminal as follows: 0 = Connected 1 = Disconnected $1-255_{10}$= Queue-depth (SF.GMC only)	–
TC.RAT	7	Specifies the type-ahead buffer as follows: 0 = 1-character type-ahead 1 = 36-character type-ahead	SET /TYPE-AHEAD=TTnn:
TC.RGS	126	Specifies that the terminal supports ReGIS instructions.	SET /REGIS=TTnn.

Table 2–8 (Cont.): Terminal Characteristics for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (If Asserted)	MCR Command
TC.RSP	3	Specifies receiver speed (bps).	SET /SPEED=TTnn:rcv:xmit
TC.SCP	12	Specifies that the terminal is a scope (CRT).	SET /CRT=TTnn:
TC.SFC	131	Specifies that the terminal supports soft character set.	SET /SOFT=TTnn:
TC.SLV	50	Indicates that no unsolicited input is accepted.	SET /SLAVE=TTnn:
TC.SMR	25	Specifies that uppercase conversion disabled.	SET /LOWER=TTnn:
TC.SSC	142	Specifies terminal management switch characters. These characteristics cause a switch from normal mode to terminal management mode.	–
TC.SXL	150	Indicates that the printer supports sixel graphics.	–
TC.TBF	71	Specifies the type-ahead buffer count obtained by SF.GMC. The count is cleared by SF.SMC.	–
TC.TBM	101	Specifies type-ahead buffer mode as follows: 0 = Task type-ahead 1 = CLI type-ahead	SET /SERIAL=TTnn:
TC.TBS	100	Indicates type-ahead buffer size ($0–255_{10}$=x) (RSX–11M–PLUS I/D systems only).	SET /TYPE-AHEAD=TTnn:x
TC.TLC	130	Indicates that CLI gets CTRL/C notification.	SET /CTRLC=TTnn:
TC.TMM	143	Indicates that characteristics are in terminal management mode. The bit is set when switch characters have been detected and terminal management mode is active. The bit is cleared by QIO$ SF.SMC. The mode is indicated as follows: 1 = In terminal management mode. 0 = Exit terminal management mode.	–

Table 2–8 (Cont.): Terminal Characteristics for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (If Asserted)	MCR Command
TC.TSY	144	Specifies output flow control. Allows input XON or XOFF to function. XOFF prevents output from the terminal as follows: 0 = XON/XOFF ignored 1 = Default; process XON/XOFF	SET /TTSYNC=TTnn:
TC.TTP	10	Indicates the terminal type ($0–255_{10}$=x.)	SET /X=TTnn: SET /TERM=TTnn:x
TC.VFL	14	Sends four fill characters after line feed for vertical forms control.	SET /VFILL=TTnn:
TC.WID[2]	1	Specifies the page width ($1–255_{10}$=x.)	SET /BUF=TTnn:x
TC.XSP	4	Specifies the transmitter speed (bps).	SET /SPEED=TTnn:rcv:xmit
TC.8BC	67	Passes 8 bits on input, even if not binary input mode (TC.BIN).	SET /EBC=TTnn:

[2]Unsolicited input that fills the buffer before a terminator is received is possibly invalid. When this happens, the driver discards the input by simulating a CTRL/U and by echoing ^U.

In Table 2–9, the octal values 0 to 177 are reserved by DIGITAL. Values 200 to 377 are available for customer use to define non-DIGITAL terminals. The implicit characteristics shown are set by the driver. Values not shown are not automatically set by the driver. An "unknown" terminal type has no implicit characteristics.

Table 2–9: Bit TC.TTP (Terminal Type) Values Set by SF.SMC and Returned by SF.GMC

Octal Value	Symbol	Terminal Type	Implicit Characters						
			TC.LPP	TC.WID	TC.HFF	TC.HHT	TC.HFL	TC.VFL	TC.TTP
0	T.UNK0	Unknown							
1	T.AS33	ASR33	66	72			1		
2	T.KS33	KSR33	66	72			1		
3	T.AS35	ASR35	66	72			1		
4	T.L30S	LA30S	66	80			7		
5	T.L30P	LA30P	66	80					
6	T.LA36	LA36	66	132					
7	T.VT05	VT05	20	72		1		1	1
10	T.VT50	VT50	12	80		1			1

Table 2–9 (Cont.): Bit TC.TTP (Terminal Type) Values Set by SF.SMC and Returned by SF.GMC

Octal Value	Symbol	Terminal Type	Implicit Characters						
			TC.LPP	TC.WID	TC.HFF	TC.HHT	TC.HFL	TC.VFL	TC.TTP
11	T.VT52	VT52	24	80		1			1
12	T.VT55	VT55	24	80		1			1
13	T.VT61	VT61	24	80		1			1
14	T.L180	LA180S	66	132		1			
15	T.V100	VT100	24	80		1			1
16	T.L120	LA120	66	132	1	1			
20	T.LA12	LA12	66	132	1	1			
21	T.L100	LA100	66	132	1	1			
22	T.LA34	LA34	66	132		1			
23	T.LA38	LA38	66	132		1			
24	T.V101	VT101	24	80		1			1
25	T.V102	VT102	24	80		1			1
26	T.V105	VT105	24	80		1			1
27	T.V125	VT125	24	80		1			1
30	T.V131	VT131	24	80		1			1
31	T.V132	VT132	24	80		1			1
32	T.LA50	LA50	66	80	1	1			
33	T.LQP1	LQP01	66	132	1	1			
34	T.LQP2	LQP02	66	132	1	1			
35	T.PC3X	PC3XX	24	80					
36	T.V2XX	VT2XX	24	80					
37	T.LN03	LN03	66	132	1	1			
40	T.DTC1	DTC01	66	132					
41	T.L210	LA210	66	132					
42	T.LQP3	LQP03	66	132					
43	T.LA75	LA75	66	80	1	1			
44	T.L2XX	LA2XX	66	132					

2.4.15.1 Characteristic Bit Special Information

The following bits have special, additional information:

- TC.HLD—Effective for VT5x and VT61 only.
- TC.RSP, TC.XSP, and TC.ASP—The list of baud rates in bps and valid MCR SET /SPEED or SET /REMOTE values that may be set is as follows:

TC.ASP, TC.RSP, or TC.XSP Value	Baud Rate (in bps) and Valid MCR SET Values
S.0	(Disabled)
S.50	50 (Baudot codes are not supported)
S.75	75
S.110	110
S.134	134
S.150	150
S.200	200
S.300	300
S.600	600
S.1200	1200
S.1800	1800
S.2000	2000
S.2400	2400
S.3600	3600
S.4800	4800
S.7200	7200
S.9600	9600
S.EXTA (DH11 external speed A)	
S.EXTB (DH11 external speed B)	
S.19.2	19200 (Not available on DZQ11 or DZV11)

Speed can be set only on the following controllers:

- DH11/DZ11
- DHU11/DHV11/DHQ11
- CXA16/CXB16
- CXY08

DZV11 and DZQ11 transmitter and receiver speeds must be equal (no split baud rates are permitted). Only one value may be specified for the remote answer speed. This value applies to both the transmitter and the receiver.

- TC.TTP—When the terminal driver reads this bit, the driver sets implicit values for terminal characteristics TC.LPP, TC.WID, TC.HFF, TC.HFL, TC.HHT, TC.VFL, and TC.SCP, as shown in Table 2–9. You can change (override) these values by subsequent IO.SMC requests. In addition, the terminal driver uses TC.TTP to determine cursor positioning commands, as appropriate.
- TC.CTS—Returns the present suspend (CTRL/S), resume (CTRL/Q), or suppress (CTRL/O) state set via the SF.SMC function. Values returned are as follows:

Value Returned	State
0	Resume (CTRL/Q)
1	Suspend (CTRL/S)
2	Suppress (CTRL/O)
3	Both suppress and suspend

When a value of 0 is used with the SF.SMC function, the suspend state is cleared; a value of 1 selects the suspend state.

- TC.TBF—Returns the number of unprocessed characters in the type-ahead buffer for the specified terminal. This allows tasks to determine whether any characters were typed that did not require AST processing. In addition, you can use the value returned to read the exact number of characters typed, rather than a typical value of 80_{10} or 132_{10} characters for the terminal. Please note the following three items when attempting to use the number returned by TC.TBF:

  1. The task must attach the terminal to receive characters from the type-ahead buffer.
  2. The maximum capacity of the type-ahead buffer is 255_{10} characters for RSX–11M–PLUS and Micro/RSX systems.
  3. Using TC.TBF in an SF.SMC function flushes the type-ahead buffer.

2.4.16 QIO$C SF.SMC—Set Multiple Characteristics

The QIO$C SF.SMC macro enables a task to set and reset the characteristics of a terminal. SF.SMC is the inverse function of SF.GMC (Get Multiple Characteristics).

Table 2–8 notes the terminal characteristics for both the SF.SMC and the SF.GMC functions.

If the characteristic-name is TC.TTP (terminal type), the octal value that corresponds to the terminal type can have any one of the values listed in Table 2–9.

A nonprivileged task can issue an SF.SMC request for its own terminal (TI) only. A privileged task can issue SF.SMC to any terminal.

Terminal output can be suspended or resumed (simulated CTRL/S and CTRL/Q, respectively) by specifying an appropriate value for TC.CTS. A value of 0 resumes output and a value of 1 suspends output. Specifying any value for TC.TBF flushes (clears) the type-ahead buffer (forces the type-ahead buffer count to 0).

For SF.SMC, the contents of the I/O Status Block (IOSB) is different from the contents of the IOSB described in Chapter 1. The first word of the status block is the same as that in Chapter 1. However, the second word is not the same. For SF.GMC or SF.SMC the second word contains the number of bytes in the specified user buffer that were sucessfully processed. For example, if you have a characteristic in the buffer that caused an error, ISB+2 (the second word) will contain the offset to the characteristic.

Format

QIO$C SF.SMC,lun,[efn],[pri],[isb],[ast], <stadd,size>

Parameters

lun

Specifies the logical unit number of the associated physical device unit to be accessed by the I/O request. For more information refer to Chapter 1.

efn

Specifies the number of the event flag to be associated with the QIO$ operation. For more information refer to Chapter 1.

pri

Makes this QIO$ macro compatible with RSX-11D. Use a value of 0 or a null for this parameter.

isb

Specifies the address of the I/O status block (I/O status doubleword) associated with the I/O request. For more information refer to Chapter 1; however, the IOSB used for SF.SMC is different from the IOSB described in Chapter 1.

ast

Allows you to interrupt your task to execute special code upon completion of this I/O request by specifying ast. When this I/O request completes, control branches to the address specified by ast at the software priority of the requesting task. Omit ast or specify 0 to omit asynchronous system trap (AST) processing.

stadd

Specifies the starting address of a buffer of length "size" bytes. The address must be word aligned for SF.SMC. Except for the characteristics TC.MHU, TC.SSC, TC.OOB, and TC.MAP, each word in the buffer has the following form:

```
.BYTE characteristic-name
.BYTE value
```

characteristic-name
Specifies one of the symbolic bit names given in Table 2–8.

value
Specifies either 0 (to clear a given characteristic) or 1 (to set a characteristic).

size
Specifies the size of the stadd data buffer in bytes. The specified size must be greater than 0 and less than or equal to 8128 bytes. The buffer must be within the task's address space. For SF.SMC, size is an even value.

2.4.16.1 Processing for TC.MAP, TC.MHU, TC.SSC, and TC.OOB

The characteristics, TC.MAP, TC.MHU, TC.SSC, and TC.OOB, require special processing and buffers. The buffers are described in the following list and have the following form:

```
.BYTE characteristic name
.BYTE reserved
.WORD ...
```

TC.MHU This characteristic declares a modem hang-up AST. The buffer required for TC.MHU is shown in Figure 2–3. The buffer must contain the address of an AST that is activated when the terminal driver detects that the carrier has been lost. A zero in word 2 (AST address) clears this characteristic.

The buffer has the format shown in Figure 2–3.

TC.SSC The characteristic TC.SSC defines or redefines terminal switch characters. The buffer required for TC.SSC is shown in Figure 2–4. The terminal must be attached before you set this characteristic.

The buffer has the format shown in Figure 2–4.

When the AST address is 0, the switch characters are disabled.

If the terminal is in terminal management mode, both CTRL/C and switch characters are treated as normal data. If the terminal is not currently in terminal management mode and switch characters have been enabled, the terminal driver compares the input characters against the specified switch characters. If there is a match, the terminal driver cancels any pending read with a status of IS.TMM, flushes the type-ahead buffer, executes the specified AST, and sets the terminal in terminal management mode.

TC.OOB The characteristic TC.OOB defines the out-of-band (OOB) character set for the particular terminal. With this characteristic you can specify certain control characters as OOB. To use TC.OOB, the task must attach the terminal and set up the TC.OOB characteristic. After TC.OOB is set, and you enter a specified OOB character at the terminal, the character causes an AST and the typed-in character is on the stack. You specify the AST address when you set up the OOB characteristic.

Additionally, you can declare any of the OOB characters as a "clear OOB character." If the character is declared to be "clear," it clears the type-ahead buffer and terminates a pending read with a status of IS.OOB. Any character that is not a "clear" can be specified as an "include character." Such a character is included in the normal input stream. "Clear OOB" may not be declared as "include."

The buffer required for TC.OOB is shown in Figure 2–5. The terminal must be attached before you set this characteristic.

Note the following items before using TC.OOB:

- Because all OOB are either HELLO or CLEAR, one set of bit masks may be used for both. A 0-bit mask is a CLEAR. A 1-bit mask is a HELLO.
- Characters that are CLEAR OOB cannot also be used for INCLUDE OOB.
- To add a character to the OOB set, all the characters must be defined, not just the one to be added.

The buffer has the format shown in Figure 2–5.

Figure 2–3: Buffer Required for TC.MHU

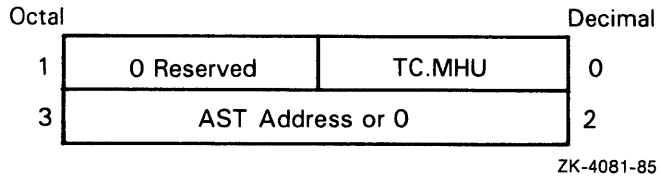


Figure 2–4: Buffer Required for TC.SSC

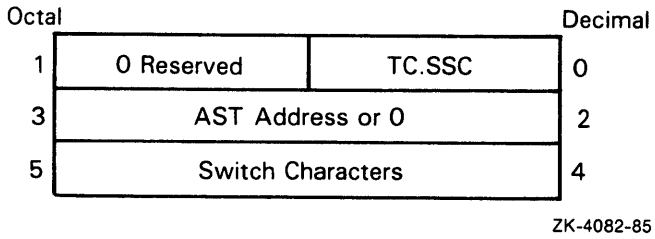
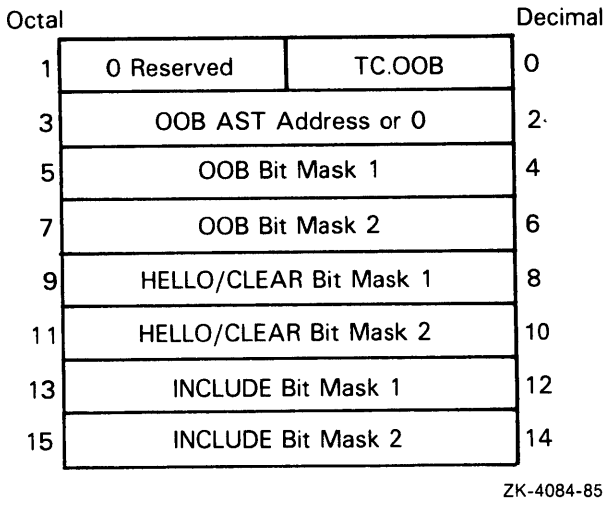


Figure 2–5: Buffer Required for TC.OOB



2.4.16.2 Side Effects of Setting Characteristics

Certain terminal characteristics that a task may set or that an operator may set using MCR or DCL commands can have undesirable side effects. In particular, the characteristics hold screen (TC.HLD), disable lowercase-to-uppercase conversion (TC.SMR), and set switch characters (TC.SSC) can have some undesirable or unexpected side effects. Their effects are described as follows:

TC.HLD Unexpected behavior can result from a terminal in the hold-screen mode if its reception rate is much greater than its transmission rate. (The DHV11 supports split baud rates.) When it is in the hold-screen mode, the terminal automatically sends a CTRL/S when an output stream is received and the screen is nearly full. Output is resumed—another screenfull—when you type SHIFT/SCROLL (the terminal generates CTRL/Q). Thus, no output is lost as a result of scrolling off the screen before you can read it. However, if the terminal's transmission rate is far below its reception rate, some unread output may scroll out of sight before the CTRL/S can be transmitted.

Note that some terminals and interfaces are hardware buffered. This can cause obscure timing problems for tasks that attempt to invoke the hold-screen mode.

TC.SMR If this characteristic is asserted (lowercase-to-uppercase conversion is disabled), octal characters 175 and 176 are interpreted as "right brace (})" and "tilde (~)," respectively. If TC.SMR is not asserted, these characters are interpreted as an ALTMODE (that is, they function as line terminators that do not advance the cursor to a new line).

TC.SSC Setting switch characters disables the normal function of CTRL/C in that it becomes a normal data character. After typing switch characters and entering terminal management mode, switch characters are normal data characters until the terminal driver exits terminal management mode.

After you have entered the first switch character, the terminal driver must wait for the second one before entering terminal management mode. If the second character is not the second switch character, the terminal driver treats both entered characters as normal data characters. Any character or combination of characters entered after the two switch characters is considered data characters.

It is advisable to specify nonordinary characters as switch characters, for example, non-system-specific CTRL/x combinations.

2.5 Status Returns

Most operating system error and status codes that are returned are byte values in the status word. For example, the value for IS.SUC is 1, which is in the low byte in the first status word. However, IS.CC, IS.CR, IS.ESC, and IS.ESQ are values in the first word of the status block. When any of these codes are returned, the low byte indicates successful completion, and the high byte shows what type of completion occurred.

To test for one of these word-value return codes, first test the low byte of the first word of the IOSB for the value IS.SUC. Then, test the full word for IS.CC, IS.CR, IS.ESC, or IS.ESQ. (If the full word is equal to IS.SUC, then its high byte is 0, indicating byte-count termination of the read.)

The "error" return IE.EOF may be considered a successful read, because the characters returned to the task's buffer can be terminated by a CTRL/Z character.

The SF.GMC and SF.SMC functions, as described in Sections 2.4.15 and 2.4.16, return the SE.xxx codes. When any of these codes are returned, the low byte in the first word in the IOSB contains IE.ABO. The second word in the IOSB word contains an offset (starting from 0) to the byte in error in the QIO's stadd buffer.

Table 2–10 lists error and status conditions that are returned by the terminal driver to the IOSB.

Table 2–10: Terminal Status Returns

Code	Reason
IE.ABO	**Operation aborted** The specified I/O operation was canceled by IO.KIL while in progress or while in the I/O queue. The second word of the IOSB indicates the number of bytes that were put in the buffer before the kill was effected.
IE.BAD	**Bad parameter** The size of the buffer is too large.

Table 2–10 (Cont.): Terminal Status Returns

Code	Reason
IE.BCC	**Framing error** A framing error was hardware detected and returned by the controller. All characters up to (but not including) the erroneous character are in the buffer. This condition can occur if you press the BREAK key on some terminals or if there are hardware problems.
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached by the issuing task. IE.DAA indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task. The subfunction bits TF.AST or TF.ESQ have no effect if IO.ATT specified them.
IE.DAO	**Data overrun error** A data overrun error was hardware detected and returned by the controller. All characters up to (but not including) the erroneous character are in the buffer. This error occurs when a hardware failure or incompatibility causes characters to be received by the controller faster than they can be processed (that is, when an incorrect serial I/O baud rate or format exists).
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	**Device not ready** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: • A timeout occurred on the physical device unit. (That is, an interrupt was lost.) • An attempt was made to perform a function on a remote DHV11 or DZV11 line without carrier present.
IE.EOF	**Successful completion on a read with end-of-file** The line of input read from the terminal was terminated with the end-of-file character CTRL/Z. The second word of the IOSB contains the number of bytes read before CTRL/Z was seen. The input buffer contains the bytes read.
IE.IES	**Invalid escape sequence** An escape sequence was started, but escape-sequence syntax was violated before the sequence was completed (see Section 2.7). The character causing the violation is the last character in the buffer.
IE.IFC	**Illegal function code** A function code specified in an I/O request was invalid for terminals, or the function code specified was a system generation option not selected for this system.

Table 2–10 (Cont.): Terminal Status Returns

Code	Reason
IE.NCB	**Buffer allocation failure** System dynamic storage has been depleted, resulting in insufficient space available to allocate an intermediate buffer for an input request or an asynchronous system trap (AST) block for an attach request.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration. The physical device unit could have been configured off line.
IE.PES	**Partial escape sequence** An escape sequence was started, but read-buffer space was exhausted before the sequence was completed. See Section 2.7.
IE.PRI	**Privilege violation** A nonprivileged task issued an IO.WBT to a terminal other than its TI or directed an SF.SMC to a terminal other than its TI. This status return is also returned when a nonprivileged task attempts to set its privilege bit.
IE.SPC	**Illegal address space** One or more of the following conditions may have occurred: • The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. • You specified a byte count of 0. • You specified an odd or 0 AST address. • You specified TF.XCC and ast2 in the same QIO$ request.
IE.VER	**Character parity error** A parity error was hardware detected and returned by the controller. All characters up to (but not including) the erroneous character are in the buffer.
IS.CC	**Successful completion on a read** The line of input read from the terminal was terminated by a CTRL/C. The input buffer contains the bytes read.
IS.CR	**Successful completion on a read** The line of input read from the terminal was terminated by a carriage return. The input buffer contains the bytes read.
IS.ESC	**Successful completion on a read** The line of input read from the terminal was terminated by an ALTMODE character. The input buffer contains the bytes read.

Table 2–10 (Cont.): Terminal Status Returns

Code	Reason
IS.ESQ	**Successful completion on a read** The line of input read from the terminal was terminated by an escape sequence. The input buffer contains the bytes read and the escape sequence.
IS.PND	**I/O request pending** The operation specified in the QIO$ directive has not yet been executed. The IOSB is filled with zeros.
IS.SUC	**Successful completion** The operation specified in the QIO$ directive was completed successfully. If the operation involved reading or writing, you can examine the second word of the IOSB to determine the number of bytes processed. The input buffer contains those bytes.
IS.TMO	**Successful completion on a read** The line of input read from the terminal was terminated by a timeout. (TF.TMO was set and the specified time interval was exceeded.) The input buffer contains the bytes read.
SE.BIN	An invalid value for a binary characteristic was used in SF.SMC.
SE.FIX	An attempt was made to change a fixed characteristic in a SF.SMC subfunction request. (For example, an attempt was made to change the unit number.)
SE.IAA	An invalid AST address was specified.
SE.NAT	The terminal is not attached.
SE.NIH	A terminal characteristic other than those listed in Table 2–8 was named in an SF.GMC or SF.SMC request.
SE.NSC	An attempt was made to change a characteristic that cannot be set. This error can occur when an attempt is made to make a local-only line a remote line when the controller does not support remote lines.
SE.SPD	The new speed specified in an SF.SMC subfunction request was not valid for the controller associated with the specified terminal.
SE.UPN	There is not enough pool space for the terminal driver to allocate buffer space.
SE.VAL	The new value specified in an SF.SMC request for the TC.TTP terminal characteristic was not one of those listed in Table 2–8.

2.6 Control Characters and Special Keys

This section describes the meanings of special terminal control characters and keys for both operating systems. Note that the driver does not recognize control characters and special keys during a Read All request (IO.RAL) or a Read with Special Terminators (IO.RST).

2.6.1 Control Characters

A control character is input from a terminal by holding the CTRL key down while pressing one other key. Three of the control characters described in Table 2–11, CTRL/R, CTRL/U, and CTRL/Z, are echoed on the terminal as ^R, ^U, and ^Z, respectively.

Table 2–11: Terminal Control Characters

Character	Meaning
CTRL/C	Pressing CTRL/C causes unsolicited input on that terminal to be directed to a command line interpreter (CLI), such as MCR. If CTRL/C abort is enabled, CTRL/C aborts tasks active at the terminal. (For this text, the assumption is that MCR is the CLI in use, although the terminal driver responds to other CLIs in a similar manner.) The "MCR> " prompt is echoed when the terminal driver is ready to accept an unsolicited MCR command line for input. When the unsolicited input is terminated, the command line is passed to MCR. If the last character typed on the terminal was a CTRL/S (suspend output), CTRL/C restarts suspended output and directs subsequent input to MCR. If the hold-screen mode system generation option has been selected and the terminal is a VT5x or VT61 in hold-screen mode, typing a CTRL/C removes the terminal from hold-screen mode. CTRL/C characters can also be directed to a task if the task has attached a terminal and has specified an unsolicited input-character AST (see Section 2.4.3). CTRL/C characters are also passed to a task if you specify a TF.RPT, IO.RAL!TF.RPT or IO.RST function, or if the task has set switch characters for the terminal. **Note** If the terminal driver receives a CTRL/C character during a read operation (except during a read-pass-all operation, during a read with special terminators operation, or when the pass-through terminal characteristic (TC.PTH) has been set), the read operation is terminated, the type-ahead buffer is cleared, and an IS.CC status code is returned to the task.
CTRL/I	CTRL/I or TAB characters initiate a horizontal tab, and the terminal spaces to the next tab stop. Tabs at every eighth character position are simulated by the terminal driver. CTRL/I or TAB have no special function if IO.RAL, IO.RST, TF.RAL, TF.RST, or TF.RPT is enabled. That is, the TAB behaves as an ordinary character.
CTRL/J	CTRL/J is equivalent to a LINE FEED character. CTRL/J has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST, or TF.RPT is enabled. That is, it behaves as an ordinary character.
CTRL/K	CTRL/K initiates a vertical tab, and the terminal tabs to the next vertical tab stop. For a CRT terminal, four LINE FEEDs are output. CTRL/K has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST, or TF.RPT are enabled. That is, it behaves as an ordinary character.

Table 2–11 (Cont.): Terminal Control Characters

Character	Meaning
CTRL/L	CTRL/L initiates a form feed. If the terminal has hardware form-feed support, the driver echoes ^L. Otherwise, the driver simulates the form feed by outputting enough line-feed characters to advance the next character position to the top of the next page. If a CRT terminal is in use, four line feeds are output. CTRL/L has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST, or TF.RPT are enabled. That is, it behaves as an ordinary character.
CTRL/M	CTRL/M is equivalent to a carriage return character (see Section 2.6.2). CTRL/M has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST, or TF.RPT are enabled. That is, it behaves as an ordinary character.
CTRL/O	CTRL/O suppresses terminal output except if IO.RAL or TF.RAL is enabled or the pass-through terminal characteristic (TC.PTH) has been set. For attached terminals, CTRL/O remains in effect (output is suppressed) until one of the following occurs: • The terminal is detached. • Another CTRL/O character is pressed. • An IO.CCO or IO.WBT function is issued. • Input is entered. • IO.RPR is issued at the terminal. For unattached terminals, CTRL/O suppresses output for only the current output buffer (typically one line).
CTRL/Q	CTRL/Q resumes terminal output previously suspended by CTRL/S except if IO.RAL or TF.RAL is enabled. This applies only to terminals for which TC.TSY is enabled (XON/XOFF are processed). You can enable TTSYNC with the MCR command SET /TTSYNC=TTnn or by setting the TC.TSY terminal characteristic bit.
CTRL/R	CTRL/R functions as a normal character if TF.RNF, IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT are enabled. Otherwise, CTRL/R results in a carriage return and a line feed being echoed, followed by the incomplete (unprocessed) input line. Any tabs that were input are expanded and the effect of anything deleted is shown. On hardcopy terminals, CTRL/R allows you to verify the effect of a tab or a delete, or both, in an input line. CTRL/R is also useful for CRT terminals when the CRT delete system generation option has been selected (see Section 2.6.2). For example, after deleting the leftmost character on the second displayed line of a wrapped input line, the cursor does not move to the right of the first displayed line. In this case, CTRL/R brings the input line and the cursor back together again.
CTRL/S	CTRL/S suspends terminal output except if IO.RAL or TF.RAL is enabled. (Output can be resumed by typing CTRL/Q or CTRL/C.) This applies only to terminals for which TTSYNC is enabled. You can enable TTSYNC with the MCR command SET /TTSYNC=TTnn or by setting the TC.TSY terminal characteristic bit.

Table 2–11 (Cont.): Terminal Control Characters

Character	Meaning
CTRL/U	CTRL/U functions as a normal character if TF.RNF, IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT are enabled. Otherwise, pressing CTRL/U before typing a line terminator deletes previously typed characters back to the beginning of the line. The system echoes this character as ^U followed by a carriage return and a line feed.
CTRL/X	CTRL/X is treated as a normal character if IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT is enabled. Otherwise, this character clears the type-ahead buffer.
CTRL/Z	CTRL/Z is treated as a normal character if IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT are enabled. Otherwise, CTRL/Z indicates an end-of-file (EOF) for the current terminal input. It notifies MAC, PIP, TKB, and other system tasks that terminal input is complete, allowing the task to exit. The system echoes this character as ^Z, followed by a carriage return and a line feed.

2.6.2 Special Keys

The ESC, RETURN, and DELETE (or RUBOUT) keys have special significance for terminal input. A line can be terminated by the ESC (or ALT) key, RETURN key, or the CTRL/Z characters, or by completely filling the input buffer (that is, by exhausting the byte count before a line terminator is typed). The standard buffer size for a terminal can be determined for a task by issuing a Get LUN Information system directive and by examining Word 5 of the buffer. An operator can obtain the same information with the MCR command SET /BUF=TI.

Table 2–12 describes the special significance of the ESC, RETURN, and DELETE (or RUBOUT) keys.

Table 2–12: Special Terminal Keys

Key	Meaning
ESC	ESC (the escape key) functions as a normal character if IO.RAL, TF.RAL, or TF.RPT are enabled. Otherwise, if escape sequences are not recognized, pressing the ESC or ALT (the ALTMODE key on some terminals) keys notifies the terminal driver that there is no further input on the current line. This line terminator allows further input on the same line, because the carriage or cursor is not returned to the first column position. If escape sequences are recognized, ESC signals the beginning of an escape sequence. (See Section 2.7.)
RETURN	Return functions as a normal character if IO.RAL, TF.RAL, or TF.RPT are enabled. Otherwise, pressing the RETURN key terminates the current line and causes the carriage or cursor to return to the first column on the next line.

Table 2-12 (Cont.): Special Terminal Keys

Key	Meaning

DELETE (or RUBOUT)	DELETE or RUBOUT functions as a normal character if TF.RNF (read no filter) is enabled. Otherwise, pressing the DELETE (or RUBOUT) key deletes the last character typed on an input line. Only characters typed since the last line terminator may be deleted. Several characters can be deleted in sequence by pressing the DELETE or RUBOUT keys successively.
	For example, on a printing terminal, the first DELETE (or RUBOUT) echoes a backslash (\) followed by the character that has been deleted, even if the terminal is in the no-echo mode. Subsequent DELETES (or RUBOUTS) cause only the deleted character to be echoed. The next character typed that is not a DELETE or RUBOUT causes another backslash to be printed, which is followed by the new character. The non-RUBOUT character is not echoed if the terminal is in the no-echo mode; however, a backslash is echoed in response to the first non-RUBOUT character. The following example illustrates rubbing out ABC and then typing CBA:
	`ABC\CBA\CBA`
	The second backslash is not displayed if a line terminator is typed after rubbing out the characters on a line, as shown in the following example:
	`ABC\CBA`
	At system generation time, the "CRT rubout" feature can be selected. This feature applies to a terminal only after a SET MCR directive has been issued as follows:
	`SET /CRT=TI:`
	If the CRT DELETE (or RUBOUT) feature was selected, pressing the DELETE (or RUBOUT) key causes the last typed character (if any) to be removed from the incomplete input line and a backspace-space-backspace sequence of characters for that terminal is echoed. If the last typed character was a tab, enough backspaces are issued to move the cursor to the character position before the tab was typed. If a long input line was split, or "wrapped," by the automatic-carriage-return option, and a DELETE (or RUBOUT) erases the last character of a previous line, the cursor is not moved to the previous line. Your task must use CTRL/R to resynchronize the current display with the contents of the incomplete input line.

2.7 Escape Sequences

Escape sequences are strings of two or more characters beginning with an escape character. Some terminals generate an escape sequence when a special key is pressed (for example, the FCN key on the VT61). On any terminal, an escape sequence may be generated manually by pressing the ESC key followed by the appropriate characters.

Escape sequences provide a way to pass input to a task without interpretation by the operating system. This could be done with a number of read-all functions, but escape sequences allow input to be read with IO.RLB requests.

2.7.1 Definition of Escape Sequence Format

The format of an escape sequence defined by American National Standards Institute's X 3.41 (1974) and used in the VT100 is shown next.

Format

ESC . . . F

Parameters

ESC

Specifies the introduced control character (033_8) that is named escape.

. . .

Specifies the intermediate bit combinations that may or may not be present. These characters are bit combination 40_8 to 57_8 inclusive in both 7- and 8-bit environments.

F

Specifies the final character. F characters are bit combinations 60_8 to 176_8 inclusive in escape sequences in both 7- and 8-bit environments.

The occurrence of a character in the inclusive range 0 to 37_8 is technically an error condition. However, the recovery from the error occurs by immediately executing the function specified by the character and then by continuing to execute the escape sequence. The exceptions to continuing the escape sequence execution are as follows:

- The character ESC occurs, aborting the current escape sequence. A new escape sequence, starting with the ESC just received, begins.
- The character CTRL/X (30_8) or the character CTRL/Z (32_8) occurs, aborting the current escape sequence. This is the case with any control character.

There are five exceptions to this general syntax definition; these exceptions are discussed in Section 2.7.5.

2.7.2 Prerequisites

There are prerequisites that must be satisfied before escape sequences can be received by a task. First, the terminal must be declared capable of generating escape sequences. This may be done using the DCL command SET as follows:

```
$ SET TERM/ESCAPE
```

After the preceding prerequisite is satisfied, one of the following prerequisites must be met:

- You must attach the terminal with IO.ATT!TF.ESQ.

- You must use the TF.RES modifier with the IO.EIO!TF.RLB function.

Note

The second method will enable escape character recognition for only the duration of the read function.

If these prerequisites are not satisfied, the ESC character is treated as a line terminator. If these prerequisites are satisfied, your task may use CTRL/SHIFT/O (017_8) as an ALTMODE character. However, this character does not act as an ALTMODE from a terminal that cannot generate escape sequences.

An ALTMODE is a line terminator that does not cause the cursor to advance to a new line. On terminals that cannot generate escape sequences, the ESC key acts as an ALTMODE. Characters 175 and 176 also function as ALTMODEs if the terminal has not been declared lowercase (DCL command SET TERM/LOWERCASE).

2.7.3 Characteristics

Escape sequences always act as line terminators. That is, an input buffer may contain other characters that are not part of an escape sequence, but an escape sequence always comprises the last characters in the buffer.

Escape sequences are not echoed. However, if a non-CRT delete sequence is in progress, it is closed with a backslash (\ ) when an escape sequence is begun.

Escape sequences are not recognized in unsolicited input streams or in a read all function (subfunction bit TF.RAL).

2.7.4 Escape Sequence Syntax Violations

A violation of the syntax defined in Section 2.7.1 causes the driver to abandon the escape sequence and to return an error (IE.IES).

2.7.4.1 Delete or Rubout (177)

The character DELETE or RUBOUT is not legal within an escape sequence. Pressing the DELETE or RUBOUT key at any point within an escape sequence causes the entire sequence to be abandoned and deleted from the input buffer. Therefore, use DELETE or RUBOUT to abandon an escape sequence, if desired, once you have begun it.

2.7.4.2 Control Characters (0 to 037_8)

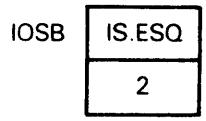
The reception of any characters other than four characters in the range 0 to 037_8 is a syntax violation that terminates the read with an error (IE.IES).

The four control characters that are allowed are: CTRL/Q, CTRL/S, CTRL/X, and CTRL/O. These characters are handled normally by the operating system even when an escape sequence is in progress. For example, entering the following command provides an I/O status block (IOSB) of line termination as shown in Figure 2–6:

```
ESC CTRL/S A
```

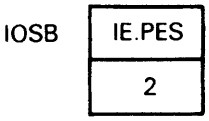
This command also provides the additional effect of turning off the output stream.

Figure 2–6: I/O Status Block Line Termination



ZK-5879-HC

Figure 2–7: I/O Status Block for Partial Escape Sequence



ZK-5880-HC

2.7.4.3 Full Buffer

When an escape sequence is terminated because there is no more buffer space rather than because you typed a final character, the error IE.PES is returned. For example, after a task issues an IO.RLB with a buffer length of 2, and you type the following line, the buffer contains "ESC !", and the IOSB contains the I/O status block for partial escape sequence shown in Figure 2–7:

```
ESC ! A
```

The "A" is treated as unsolicited input.

2.7.5 Exceptions to Escape Sequence Syntax

The following five "final characters" that normally terminate an escape sequence are treated as special cases by the terminal driver for use with certain terminals:

ESC ?...
ESC O...
ESC P...
ESC Y...
ESC [...

Refer to documentation supplied with the specific terminal or terminals in use for correct use of escape sequences.

2.8 Vertical Format Control

Table 2–13 is a summary of all the characters that your task can use for vertical format control on the terminal. Any one of these characters can be specified as the value of the vfc parameter in the IO.WLB, IO.WVB, IO.WBT, IO.CCO, or IO.RPR functions.

Table 2–13: Vertical Format Control Characters

Octal Value	Character	Meaning
040	Blank	**Single Space** Outputs one line feed, prints the contents of the buffer, and outputs a carriage return. Normally, printing immediately follows the previously printed line.
060	0	**Double Space** Outputs two line feeds, prints the contents of the buffer, and outputs a carriage return. Normally, the buffer contents are printed two lines below the previously printed line.
061	1	**Page Eject** If the terminal supports form feeds, it outputs a form feed, prints the contents of the buffer, and outputs a carriage return. If the terminal does not support form feeds, the driver simulates the form-feed character by either outputting four line feeds to a CRT terminal, or by outputting enough line feeds to advance the paper to the top of the next page on a printing terminal.
053	+	**Overprint** Prints the contents of the buffer and outputs a carriage return, normally overprinting the previous line.
044	$	**Prompting Output** Outputs one line feed and prints the contents of the buffer. This mode of output is used with a terminal on which a prompting message is output and input is then read on the same line.
000	Null	**Internal Vertical Format** Prints the buffer contents without addition of vertical format control characters. In this mode, more than one line of guaranteed contiguous output can be printed for each I/O request.

All other vertical format control characters are interpreted as blanks (040).

2.9 Automatic Carriage Return

You can set individual terminals for wraparound, as desired, by using the MCR command SET /WRAP, as follows:

```
>SET /WRAP=TTxx:
```

Once you select wraparound, you can select the column at which wraparound occurs by using the MCR command SET /BUF as follows:

```
>SET /BUF=TI:n
>
```

Your task can also use the MCR command SET /BUF without an argument to display the current buffer width for a terminal as follows:

```
>SET /BUF=TI:
BUF=TI:00072.
>
```

A task can determine the buffer width by issuing a Get LUN Information directive and by examining word 5 returned in the buffer.

After the MCR command SET /BUF has been entered, typing beyond the buffer width results in a carriage return and line feed being output before the next character is echoed. Although you may have typed only one line, it is displayed on two terminal lines.

You can lose track of where you are in the input buffer if wraparound is enabled for your terminal. For example, while deleting text on a wrapped line, the cursor does not back up to the previous line. To resynchronize the cursor with the contents of the incomplete input buffer, type CTRL/R (if this option was selected during system generation).

2.10 Hard Receive Error Detection

All terminal interfaces supported by the full-duplex terminal driver are capable of detecting and flagging hard receive errors. Hard receive errors include framing errors, enable character parity error, and data overrun error.

The terminal driver handles hard receive errors as follows:

1. If a read request is being processed and the character can be processed immediately, the read request is terminated with one of the following error codes returned in the status block:

Error Code	Hard Receive Error
IE.BCC	Framing
IE.DAO	Data overrun
IE.VER	Character parity

2. If a command line is being input for a command line interpreter (CLI) task and the character can be processed immediately, a CTRL/U is simulated, ^U is echoed, and the input is terminated. No command line is sent to the task.
3. If the character would normally cause an asynchronous system trap (AST) if no error was detected, the character is ignored and no AST occurs.

4. If the character cannot be processed immediately, it is stored in the type-ahead buffer. A flag is set for the line, indicating that the last character in the type-ahead buffer has an error, disabling further storage in the type-ahead buffer. When the character is retrieved from the buffer, the appropriate action is taken, and the flag is cleared. Any characters received in the meantime are discarded, and a bell is echoed for each character.

2.11 Task Buffering of Received Characters

When task buffering received characters, characters read from the terminal are sent directly to the task's buffer. Thus, there is no need to allocate a terminal driver buffer.

Task buffering of received characters does not necessarily reduce system overhead. For example, each character must be mapped to the task's buffer. However, if terminal driver buffering was used, the system does the mapping only once for all characters to be transferred.

With the full-duplex terminal driver, output buffering is always performed.

Task buffering is overridden during checkpointing. If a task is checkpointable, a driver buffer is allocated and the task is made eligible for checkpointing by any task, regardless of priority, while the read operation is in progress. (Checkpointing occurs in this situation only when there is another task that can be made active.) Because checkpointability is controlled by the task, you retain control over this operation.

2.12 Type-Ahead Buffering

Characters received by the terminal driver are either processed immediately or stored in the type-ahead buffer. The type-ahead buffer allows characters to be temporarily stored and retrieved FIFO (first in, first out). The terminal driver uses the type-ahead buffer as follows:

1. Store in buffer

   An input character is stored in the type-ahead buffer if one or more of the following conditions are true:

   - The driver is not ready to accept the character (fork process pending or in progress).
   - There is at least one character presently in the type-ahead buffer.
   - The character input requires echo, and the output line to the terminal is presently busy outputting a character.
   - No read request is in progress, no unsolicited input AST is specified, and the terminal is either attached or slaved and attached.

   Note

   Depending on the terminal mode and the presence of a read function, read subfunctions, and an unsolicited input AST, the CTRL/C, CTRL/O, CTRL/Q, CTRL/S, and CTRL/X characters may be processed immediately and not stored in the type-ahead buffer.

   A character is not echoed when it is stored in the buffer. Echoing a character is deferred until it is retrieved from the buffer, because the read mode (for example, read-without-echo) is not known by the driver until then.

2. Retrieve from buffer

When the driver becomes ready to process input, or when a task issues a read request, it attempts to retrieve a character from the buffer. If the attempt is successful, the character is processed and echoed, if required. The driver then loops, retrieving and processing characters until either the buffer is empty, the driver becomes unable to process another character, or a read request is finished with the terminal attached.

3. Flush the buffer

   The buffer is flushed (cleared) when:

   - CTRL/C is received.
   - CTRL/X is received.
   - A clear out-of-band (OOB) character is entered.
   - Switch characters are detected.
   - The terminal becomes detached.
   - TC.TBF is written by SF.SMC.
   - **Exceptions:** CTRL/C and CTRL/X do not flush the buffer if read-pass-all or read-with-special-terminators is in effect. If the buffer becomes full, each character that cannot be entered causes a BELL character to be echoed to the terminal.

   If a character is input and echo is required, but the transmitter section is busy with an output request, the input character is held in the type-ahead buffer until output (transmitter) completion occurs.

2.13 Full-Duplex Operation

When a terminal line is in the full-duplex mode, the full-duplex driver attempts to service one read request and one write request simultaneously. The IO.ATA, IO.ATT, IO.DET, and SF.SMC functions are performed with the line in an idle state only (not executing a read or a write request).

2.14 Private Buffer Pool

The driver has a private buffer pool for intermediate input and output buffers. Whenever the driver needs dynamic memory, it first attempts to allocate a buffer in the private pool. If this fails, it attempts to allocate a buffer in the system pool. If the allocation in the system pool fails during command line input, a CTRL/U is simulated and echoed.

CLI task buffers are handled in a special way. When unsolicited input begins, a buffer is allocated, as previously described, for the command line (a string of characters, followed by an appropriate terminator character). When the input is completed, the contents of the buffer is sent directly to the CLI task if the buffer was allocated in the system pool. However, if the buffer was allocated in the driver's private pool, it must first be moved into a buffer in the system pool to provide access for the task.

2.15 Intermediate Input and Output Buffering

Input buffering for checkpointable tasks with checkpointing enabled is provided in the private pool. As each buffer becomes full, a new buffer is automatically allocated and linked to the previous buffer. The Executive then transfers characters from these buffers to the task buffer, and the terminal driver deallocates the buffers once the transfer has been completed.

If the driver fails to allocate the first input buffer, the characters are transferred directly into the task buffer. If the first buffer is successfully allocated, but a subsequent buffer allocation fails, the input request terminates with the error code IE.NOD. In this case, the IOSB contains the number of characters actually transferred to the task buffer. The task may then update the buffer pointer and byte count and reissue a read request to receive the rest of the data. The type-ahead buffer ensures that no input data is lost.

All terminal output is buffered. As many buffers as required are allocated by the terminal driver and linked to a list. If not enough buffers can be obtained for all output data, the transfer is done as a number of partial transfers, using available buffers for each partial transfer. This is transparent to the requesting task. If no buffers can be allocated, the request terminates with the error code IE.NOD.

The unconditional output buffering serves three purposes:

1. It reduces time spent at interrupt level.
2. It enables long direct memory access (DMA) transfers for DH11 controllers.
3. It enables task checkpointing during the transfer to the terminal (if all output fits in one buffer list).

2.16 Terminal-Independent Cursor Control

Terminal-independent cursor control capability is provided during system generation. The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use. I/O functions associated with cursor positioning are described in the following paragraphs.

Cursor position is specified in the vfc parameter of the IO.WLB or IO.RPR function. The parameter is interpreted simply as a vfc parameter if the high byte of the parameter is 0. However, if the parameter defines cursor position, the high byte must be nonzero, the low byte is interpreted as column number (x-coordinate), and the high byte is interpreted as line number (y-coordinate). Home position, the upper left corner of the display, is defined as (1/1). Depending on terminal type, the driver sends to the terminal cursor-positioning commands appropriate for the terminal in use that move the cursor to the specified position. If the most significant bit of the line number is set, the driver clears the display before positioning the cursor.

When defining cursor position in an IO.WLB function, you can use the TF.RCU subfunction to save the current cursor position. When included in this manner, TF.RCU causes the driver first to save the current cursor position, then to position the cursor and output the specified buffer, and, finally, to restore the cursor to the original (saved) position once the output transfer has been completed.

2.17 Terminal Interfaces

This section summarizes the characteristics of the standard communication-line interfaces supported by RSX–11M–PLUS and Micro/RSX. Refer to the DIGITAL *Terminals and Communications Handbook* or *Microcomputer Products Handbook* for additional details.

2.17.1 DH11 Asynchronous Serial Line Multiplexer

The DH11 multiplexer interfaces up to 16 asynchronous serial communication lines for terminal use. The DH11 supports programmable baud rates. Input and output baud rates may differ; the input rate may be set to 0 baud, thus effectively turning off the terminal. The DM11-BB option may be included to provide modem control for dial-up lines. These lines must be interfaced by a full-duplex modem (for example, in the United States, a Bell 103A or equivalent modem).

2.17.2 DHU11 Asynchronous Serial Line Multiplexer

The DHU11 is a 16-line, UNIBUS asynchronous multiplexer with DMA. The DHU11 supports program-selectable baud rates with the option of selecting split-speed operation. In addition, full modem control is available on all 16 lines.

2.17.3 DHQ11 Asynchronous Serial Line Multiplexer

The DHQ11 is a 16-line, Q-bus asynchronous multiplexer with DMA. The DHU11 supports program-selectable baud rates with the option of selecting split-speed operation. In addition, full modem control is available on all 16 lines.

2.17.4 DHV11 Asynchronous Serial Line Multiplexer

The DHV11 multiplexer interfaces up to eight asynchronous serial communication lines for terminal use. This multiplexer is the Q-bus version of the DHU11 UNIBUS multiplexer. The DHV11 supports programmable baud rates with the option of selecting split-speed operation. (Split-speed operation allows different transmit and receive speeds.) Also provided is modem control for full-duplex, point-to-point operation.

2.17.5 DJ11 Asynchronous Serial Line Multiplexer

The DJ11 multiplexer interfaces as many as 16 asynchronous serial lines to the PDP–11 for local terminal communications. The DJ11 does not provide a dial-up capability. Baud rates are jumper selectable.

2.17.6 DL11 Asynchronous Serial Line Interface

The DL11 supports a single asynchronous serial line and handles communication between the PDP–11 and a terminal. A number of standard baud rates are available to DL11 users. However, because the DL11 does not have an input silo, baud rates greater than 1200 baud are not recommended. Higher baud rates may cause input characters to be lost.

For hardware design reasons, a DL11 is susceptible to losing receiver-interrupt-enable in its Receiver Status Register. The disabling of the receiver interrupt bit causes the terminal to print output requests but not to respond to input (for example, the terminal does not echo input characters). The terminal driver has no mechanism for recognizing the disabling of the receiver interrupt bit. Therefore, it cannot recover. The bit must be reset with the MCR command OPEN, the console switch register, or a periodically rescheduled task.

2.17.7 DZ11 Asynchronous Serial Line Multiplexer

The DZ11 multiplexer interfaces up to eight asynchronous serial communication lines for use with terminals. It supports programmable baud rates; however, transmit and receive baud rates must be the same. The DZ11 can control a full-duplex modem in autoanswer mode.

2.17.8 DZQ11 Asynchronous Serial Line Multiplexer

The DZQ11 multiplexer interfaces up to four asynchronous serial data communication lines for use with terminals. It supports programmable baud rates with limited modem control on each line and provides both local and remote interconnection.

2.17.9 DZV11 Asynchronous Serial Line Multiplexer

The DZV11 is an asynchronous multiplexer interface that connects the Q-bus with up to four serial communication lines for use with terminals. It supports program-selectable baud rates with limited modem control on each line.

2.17.10 CXA16/CXB16 Asynchronous Multiplexers

The CXA16/CXB16 are LSI-11/Q-bus asynchronous multiplexers which provide 16 full duplex serial data-only channels for use on the Q-bus BA200-series systems.

2.17.11 CXY08 Asynchronous Multiplexer

The CXY08 is a quad-height, Q-bus asynchronous multiplexer, which provides eight full duplex serial data channels for use on the Q-bus BA200-series and DECSERVER 500 systems. The CXY08 can be used for point-to-point operation over private lines. Modem control is implemented by software in the host system.

2.18 Programming Hints

The following sections are supplied as additional general information to enhance your use of the full-duplex terminal driver.

2.18.1 Checkpointing During Terminal Input

If checkpointing during terminal input was selected as a system generation option, a checkpointable task is stopped (and therefore eligible to be checkpointed) when trying to read. Therefore, a program that issues a read function followed by a Mark Time directive does not work. The intent might be to time out the read if input is not received in a reasonable length of time. But the Mark Time directive is not issued until the read completes.

You can circumvent this behavior by disabling checkpointing for the read. This is not a desirable solution because it forces a task to remain in memory during the entire read. This defeats the purpose of selecting the checkpoint-during-terminal-input option.

2.18.2 RT02-C Control Function

Because the screen of an RT02-C Badge Reader and Data Entry Terminal holds only one line of information, special care must be taken when sending a control character (for example, vertical tab) to the RT02-C. Use the IO.WAL (write all) function for this purpose.

It is recommended that your task use read without echoing when reading a badge with the RT02-C. Use IO.RAL or IO.RNE functions, followed by the IO.WAL function, to echo the information for display.

2.18.3 Remote DL11-E, DH11, and DZ11 Lines

Before a remote line is answered, the driver clears certain terminal characteristics (see Table 2–8) that may have been set by the MCR command SET or by an SF.SMC function. The characteristics cleared are: TC.SCP, TC.ESQ, TC.HLD, TC.SMR, TC.NEC, TC.FDX, TC.HFF, TC.HHT, TC.VFL, TC.HFL, TC.TTP, TC.8BC, TC.PTH, and TC.BIN. (Clearing TC.TTP means that a terminal type of "unknown" returns in an SF.GMC request.) The TC.ACR characteristic (automatic wraparound) is set. Buffer size is set to 72.

A DZ11 remote line must be declared to be remote before the terminal driver can handle the modem.

2.18.4 Modem Support

The terminal driver supports the following modem control operations:

- Local or remote operation
- Answer speed
- Autobaud speed detection

The characteristics bit that controls local or remote operation is TC.DLU. This bit can be set with the MCR command SET /REMOTE (or SET /NOREMOTE for local operation). The DCL command SET TERMINAL REMOTE (or SET TERMINAL LOCAL) can also be used.

When there is an incoming call on a remote line, the TC.ASP characteristic determines the baud rate for the answering modem.

Split baud rates (different transmit and receive speeds) are not supported for answer speed.

The default answer speed is set during system generation. However, the answer speed can be set on line using the MCR command SET /REMOTE=TTnn:speed. The Virtual Monitor Console Routine (VMR) can also be used to set the answer speed.

The terminal driver can determine the speed of the incoming call by sampling the first input character after dial-up for the following speeds:

110 150 300 600 1200 1800 2400 4800 9600

This is called autobaud speed detection. It is an option that you can select for each line by using the MCR command SET /AUTOBAUD. When you set autobaud speed detection for a given line, the terminal driver tries to sense the baud speed of the caller when the caller's line is set to remote and a call has been received. The terminal driver detects the baud rate as you press the RETURN key (enter carriage returns) several times when you first establish the remote

connection from your terminal to the remote computer. Press the RETURN key until the default RSX prompt (>) is displayed.

The MCR command SET /AUTOBAUD sets the TC.ABD terminal characteristic.

Chapter 3
Virtual Terminal Driver

3.1 Introduction to the Virtual Terminal

The virtual terminal driver supports offspring task use of virtual terminals in RSX–11M–PLUS and Micro/RSX systems. Virtual terminals are not physical hardware devices; they are actually implemented in software through the use of data structures created by the Executive. Virtual terminals are created by the Executive when requested by parent tasks with the Create Virtual Terminal directive. Virtual terminals are useful in batch processing and other processing environments in providing noninteractive terminal I/O support for offspring tasks, thereby eliminating the need for operator intervention.

Offspring task or tasks "spawned" by or "connected" to the parent task that created the virtual terminal can perform terminal I/O operations with the virtual terminal in the same manner as with physical terminals. Virtual terminals differ from physical terminals in that they receive input from or output to a program (the parent task), rather than from a keyboard or to a display (or printer), respectively.

3.2 Get LUN Information Macro

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for virtual terminals. A setting of 1 indicates that the described characteristic is true for virtual terminals.

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	1	Terminal device
3	0	File-structured device
4	0	Single-directory device

Bit	Setting	Meaning
5	0	Sequential device
6	0	Reserved
7	0	User-mode diagnostics supported
8	0	MASSBUS device
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a Files–11 volume
15	0	Device mountable

Words 3 and 4 are undefined. Word 5 specifies the maximum byte count (that is, maximum buffer size) to which offspring requests will be truncated; this value is specified by the parent task in the Create Virtual Terminal system directive, as described in the *RSX-11M-PLUS and Micro/RSX Executive Reference Manual.*

3.3 QIO$ Macro

Table 3–1 lists the standard and device-specific functions of the QIO macro that are valid for virtual terminals.

Table 3–1: Standard and Device-Specific QIO Functions for Virtual Terminals

Format	Function
STANDARD FUNCTIONS:	
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O request
QIO$C IO.RLB,..., <stadd,size>	Read logical block
QIO$C IO.RVB,..., <stadd,size>	Read virtual block (effects IO.RLB)
QIO$C IO.WLB,..., <stadd,size,stat>	Write logical block
QIO$C IO.WVB,..., <stadd,size,stat>	Write virtual block (effects IO.WLB)

Table 3–1 (Cont.): Standard and Device-Specific QIO Functions for Virtual Terminals

Format	Function
DEVICE-SPECIFIC FUNCTIONS:	
QIO$C IO.STC,..., <cb,sw2,sw1>	Set terminal characteristics (enable/disable intermediate I/O buffering, or return I/O completion status to offspring task)
QIO$C SF.GMC,..., <stadd,size>	Get multiple characteristics
QIO$C IO.GTS,..., <stadd,size>	Get terminal support
QIO$C IO.RPR,..., <stadd,size,[tmo], pradd,prsize,vfc>	Read logical block after prompt
QIO$C SF.SMC,..., <stadd,size>	Set multiple characteristics

Parameters

size

Specifies the size of the data buffer in bytes (must be greater than 0). The buffer must be located within the addressing space of the parent or offspring task issuing the I/O request.

stadd

Specifies the starting address of the data buffer. The address must be word aligned for SF.GMC, IO.GTS, and SF.SMC; otherwise, it may be aligned on a byte boundary.

stat

Specifies the I/O completion status code, specified by the parent task, that is issued by the virtual terminal driver in response to an offspring task's read request upon successful completion.

cb

Specifies the characteristic bits to be set, selecting the following virtual terminal functions:

cb Value	Bits Set	Function
0	None	Enable intermediate buffering in the Executive pool
1	0	Return the specified virtual terminal I/O completion status to the requesting offspring task
2	1	Disable intermediate buffering
3	0 and 1	Return status for offspring write request

sw2

Specifies enable/disable intermediate I/O buffering.

sw1

Specifies the I/O completion code for I/O completion status.

Note

The sw2 and sw1 parameters are valid in the IO.STC function only when cb=1 or cb=3.

tmo

Specifies an optional timeout count (see Section 3.3.1.4).

vfc

Specifies a character for vertical format control.

pradd

Specifies the starting address of the prompt buffer.

prsize

Specifies the size of the prompt buffer in bytes. The buffer must be located within the address space of the offspring task issuing the I/O request.

3.3.1 Standard QIO Functions

The following sections describe the standard QIO funtions associated with the virtual terminal driver.

3.3.1.1 IO.ATT

The IO.ATT function can be issued by offspring tasks to attach the virtual terminal. (It is illegal for parent tasks to issue IO.ATT). Attaching a virtual terminal prevents other offspring tasks from executing I/O operations with the virtual terminal. However, parent task I/O requests are always serviced when issued.

3.3.1.2 IO.DET

The IO.DET function can be issued by offspring tasks to detach the virtual terminal, making it available for use by other offspring tasks connected to the same parent task. (It is illegal for parent tasks to issue IO.DET.)

3.3.1.3 IO.KIL

Parent and offspring tasks can issue an IO.KIL function to cancel I/O requests. An offspring task issuing IO.KIL can result in IE.ABO (operation aborted) being returned to the parent task.

3.3.1.4 IO.RLB, IO.RVB, IO.WLB, and IO.WVB

These read and write functions execute the requested I/O operations described in Chapter 2, with the following two exceptions:

1. The virtual terminal driver returns the tmo parameter of an offspring task's IO.RLB or IO.RVB request, or it returns the vfc parameter of an offspring task's IO.WLB or IO.WVB request as a stack parameter on entry to the appropriate asynchronous system trap (AST) for the parent task.
2. The virtual terminal driver returns I/O completion status to the offspring task in response to successful completion of the offspring task's IO.RLB or IO.RVB request; however, the actual I/O completion status values returned are specified for data transfers in the third parameter word of the parent task's IO.WLB or IO.WVB response, or in the second and third parameters of the parent task's IO.STC function response when no data transfer is desired.

3.3.2 Device-Specific QIO Function (IO.STC)

The IO.STC function can be issued by parent tasks to enable/disable offspring task I/O buffering in secondary pool, or the function can be used to force an appropriate I/O completion status for an offspring task read I/O request when no data transfer is desired. Both of these applications for the IO.STC function are described in the following paragraphs.

Parent tasks can use IO.STC to enable (or disable) intermediate buffering in secondary pool. Intermediate buffering, when enabled, is performed on offspring task virtual terminal read and write requests when the offspring task is checkpointable.

Thus, offspring tasks can be stopped for virtual terminal I/O and checkpointed in a manner similar to the mannner used when you use physical terminals. Whenever the virtual terminal driver determines that it should not use intermediate buffering, offspring tasks that issue terminal requests become locked in memory until I/O completion; transfers occur directly between parent task and offspring task buffers without intermediate buffering in secondary pool.

In addition to the conditions that permit intermediate buffering (when specified), one condition can disable intermediate buffering of the parent task. If the buffer size specified in the Create Virtual Terminal directive exceeds the maximum size specified at system generation time (512_{10} maximum), intermediate buffering is disabled.

The second application for IO.STC is to allow the virtual terminal driver to return an appropriate I/O completion status in response to an offspring task read request. An I/O status returned in this manner allows successful completion of the offspring task's request when the parent task determines that no data transfer is desired; this condition can occur, for example, when no data is available for input to the offspring task by the virtual terminal driver. When you use the IO.STC function in this manner, you must use the format shown next.

Format

```
QIO$C IO.STC   ,..., <cb,sw2,sw1>
```

Parameters

cb

Specifies a value of 1 to indicate that the I/O completion status returned to the offspring task is desired.

Note

If the virtual terminal is operating in full-duplex mode, a cb value of 1 returns status for an offspring read request, and a cb value of 3 returns status for an offspring write request.

sw2

Specifies the second word returned in the I/O completion status and indicates the number of bytes read upon successful completion of an offspring task's read request. However, because no data transfer actually occurs, the value specified is 0. The byte count of 0 specified in this function is legal (and desired); whereas, a byte count of 0 in write operations is illegal (and results in an error being returned to the parent task).

sw1

Specifies the status code to be returned to the offspring task by the virtual terminal driver in the first word of the I/O completion status. This value is returned in the high byte and a value of +1 is returned in the low byte of the status word. Typical values and the status that each represents are listed as follows:

Code	Value	Completion Status Indicated
IS.SUC	+1	Successful completion
IS.CR	15	Read terminated by carriage return
IS.ESC	33	Read terminated by an ALTMODE
IS.ESQ	233	Read terminated by an escape sequence

3.3.3 SF.GMC

The Get Multiple Characteristics function returns information on terminal characteristics. This function can be issued by both the parent and the offspring tasks. The virtual terminal driver returns the characteristics that were set by the previous corresponding SF.SMC request. However, only the full-duplex mode (TC.FDX) characteristic affects the operation of the virtual terminal driver. The SF.GMC function is provided only to maintain transparency to the offspring task.

Valid virtual terminal characteristics are listed in Table 3–2.

3.3.4 IO.GTS

The Get Terminal Support function returns a 4-word buffer of information that specifies which features are a part of the virtual terminal driver. The virtual terminal driver provides the IO.GTS function only to maintain transparency to the offspring task. Table 3–2 lists the options returned by the full-duplex terminal driver. Of the options listed, the virtual terminal driver returns the following:

Word 1 F1.BUF, F1.RPR, F1.UTB, and F1.VBF

Word 2 F2.SCH and F2.GCH

3.3.5 IO.RPR

The Read After Prompt (IO.RPR) function can be issued only by the offspring task. When the offspring task issues this function, the function appears to the parent task as a separate write request followed by a read request.

IO.RPR has the same effect as IO.WLB (to write a prompt to the terminal) followed by IO.RLB. However, IO.RPR differs in the following four ways from this combination of QIOs:

1. System overhead is lower because only one QIO is processed.
2. There is no "window" during which a response to the prompt may be ignored. Such a window occurs if you use IO.WLB/IO.RLB, because no read may be posted at the time the response is received.
3. If the issuing task is checkpointable, it is checkpointed during both the prompt and the read.
4. A CTRL/O that may be in effect is canceled before the prompt is written.

The third argument that you can specify to IO.RPR, the tmo argument, is required for compatibility with IAS. If supplied, it is ignored.

Subfunction bits may be ORed with IO.RPR to write the prompt as a Write All (TF.BIN) and to send XOFF after the read (TF.XOF). In addition, you can use the three Read subfunction bits (TF.RAL, TF.RNE, TF.RST) with IO.RPR.

3.3.6 SF.SMC

The SF.SMC function allows a task to set and reset the characteristics of a terminal. Both the parent and the offspring tasks may issue this function. The parent task may set virtual terminals to full-duplex operation by using the SF.SMC function with the characteristics bit TC.FDX. When in full-duplex mode, the virtual terminal driver attempts to process the offspring task's read and write requests simultaneously. To ensure that these operations are overlapped, the parent task should minimize the amount of time it spends in AST state.

Note

The virtual terminal driver defaults to half duplex mode.

Table 3–2 lists the characteristics that either the parent or the offspring task may set.

Table 3–2: Virtual Terminal Characteristics

Bit Name	Octal Value	Meaning (If Asserted)	Default Value
TC.FDX	64	Full-duplex mode	0
TC.SCP	12	Terminal is a scope	0
TC.SMR	25	Uppercase conversion disabled	0
TC.TTP	10	Terminal type	0

3.4 Status Returns

The error and status conditions listed in Tables 3–3 and 3–4 are returned by the virtual terminal driver. The SE.NIH error is returned by the SF.GMC and SF.SMC functions. For this error, the low byte of the first word in the I/O status block (IOSB) contains IE.ABO. The second word in the I/O status block contains an offset (starting at 0) pointing to the erroneous byte in the stadd buffer.

Table 3–3: Virtual Terminal Status Returns for Offspring Task Requests

Code	Reason
–	Successful completion of an offspring task read request results in an I/O completion status specified in a parent task QIO parameter being returned. Typically, the status information returned simulates a subset of I/O returns normally produced by the terminal driver described in Chapter 2.
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the IOSB indicates the number of bytes transferred on a write operation.
IE.IFC	**Invalid function code** The offspring task attempted a read or a write function and the parent task did not specify an AST address in its response to the requested I/O function, or the offspring task issued an IO.STC or other invalid function.
IE.ABO	**Request terminated** The offspring task issued IO.KIL or the parent task eliminated the virtual terminal unit.
IE.SPC	**Illegal address space** Part or all of the buffer specified for a read or write request was outside of the task's address space, or a byte count of 0 was specified.

Table 3–3 (Cont.): Virtual Terminal Status Returns for Offspring Task Requests

Code	Reason
IE.UPN	**Insufficient dynamic storage** The driver could not allocate an AST block to notify the parent task of an offspring task request, or the driver could not allocate an intermediate buffer in the Executive pool.
SE.NIH	A terminal characteristic other than those listed in Table 3–2 was specified, or an offspring task attempted to assert TC.FDX.

Table 3–4: Virtual Terminal Status Returns for Parent Task Requests

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the IOSB indicates the number of bytes transferred on a read or write operation.
IE.EOF	**End of file encountered** The IO.STC function was completed successfully.
IE.BAD	**Bad parameters** The parent task specified a buffer size that exceeded the system maximum specified at system generation time.
IE.DUN	**Device not attachable** An IO.ATT or IO.DET function was issued by the parent task.
IE.IFC	**Invalid function code** A read, write, or IO.STC function was issued without a pending offspring task request. This status can occur if the offspring task cancels a pending read or write request. This function code is also returned when IO.STC is issued to enable intermediate buffering on a virtual terminal unit whose buffer size, specified in the Create Virtual Terminal directive, exceeds the system maximum specified at system generation time.
SE.NIH	A terminal characteristic other than those listed in Table 3–2 was specified in an SF.GMC or SF.SMC request.

Chapter 4
Disk Drivers

4.1 Introduction to Disk Drivers

The RSX-11M-PLUS and Micro/RSX disk drivers support the disks summarized in Table 4-1. Subsequent sections describe these devices in greater detail.

All the disks described in this chapter are accessed in essentially the same manner. Up to eight disks of each type (except RX01, RX02, RX33, RX50, RD51, RD52, RC25, RL01, RL02, RA60, RA80, or RA81) may be connected to their respective controllers. Disks and other file-structured media are divided logically into series of 256-word blocks.

Table 4-1: Standard Disk Devices

Drive	Revolutions Per Minute	Sectors	Tracks	Cylinders	Bytes/ Drive	Decimal Blocks
RS11	1800	–	1	128	524,288	1024
RS03	3600	64[1]	1	64	524,288	1024
RS04	3600	64[1]	1	64	1,048,576	2048
RPR02	2400	10	20	200	20,480,000	40,000
RP03	2400	10	20	400	40,960,000	80,000
RM02	2400	32	5	823	67,420,160	131,680
RM03	3600	32	5	823	67,420,160	131,680
RM05	3600	32	19	823	256,196,608	500,384
RP04,RP05	3600	22	19	411	87,960,576	171,798
RP06	3600	22	19	815	174,423,040	340,670

[1] The RS03 has 64 words per sector; the RS04 has 128 words per sector.

Table 4–1 (Cont.): Standard Disk Devices

Drive	Revolutions Per Minute	Sectors	Tracks	Cylinders	Bytes/ Drive	Decimal Blocks
RP07	3600	50	32	630[2]	516,096,000	1,008,000
RM80	3600	31	14	559[2]	124,214,272	242,606
RK05	1500	12	2	200	2,457,600	4800
RL01	2400	40[3]	2	256	5,242,880	10,240
RL02	2400	40[3]	2	512	10,485,760	20,480
RK06	2400	22	3	411	13,888,512	27,126
RK07	2400	22	3	815	27,810,800	53,790
RX01	360	26[4]	1	77	256,256	494
RX02	360	26[4]	1	77	512,512	988
RA80	3600	31	14	546	121,325,568	236,964
RA81	3600	51	14	1248	456,228,864	891,072
RA60	3600	42	4	2382	204,890,112	400,176
RC25	2850	31	2	796	26,061,824	50,902
RD31	–	–	–	–	–	–
RD32	–	–	–	–	–	–
RD51	3600	16	4	306	10,027,008	19,584
RD52	Manufacturer dependent				30,9657,60	60,480
RD53	–	–	–	–	–	–
RD54	–	–	—	–	–	–
RX50	300	10	1	80	409,600	800
RX33	360	15	160	615	1,228,800	2400

[2]The RP07 and the RM80 each have two additional CE cylinders.

[3]The RL01 and RL02 each have 128 words per sector.

[4]The RX01 has 64 words per sector; the RX02 has 128 words per sector.

4.1.1 RF11/RS11 Fixed-Head Disk

The RF11 controller/RS11 fixed-head disk provides random access bulk storage. It features fast track-switching time and a redundant set of timing tracks.

4.1.2 RS03 Fixed-Head Disk

The RS03 (RH11-RH70 controller/RS03 fixed-head disk) is a fixed-head disk that offers speed and efficiency. With 64 tracks per platter and recording on one surface, the RS03 has a capacity of 262,144 words.

The RS04 (RH11-RH70 controller/RS04 fixed-head disk) is similar to the RS03 disk and interfaces to the same controller, but the RS04 provides twice the number of words per track by recording on both surfaces of the platter, and thus has twice the capacity.

The RP11 controller/RP02 or RP03 disk pack consists of 20 data surfaces and a moving read/write head. The RP03 has twice as many cylinders, and thus double the capacity of the RP02. Only an even number of words can be transferred in a read/write operation.

4.1.3 RM02/RM03/RM05/RM80 Disk Pack

The RM02/RM03, RM05, and RM80 are MASSBUS disk drives and adapters that use the existing MASSBUS controller. With a single head per surface, they provide a 1.2-Mb/s data transfer rate. PDP–11/70 systems use the RM03, RM05, and RM80 with the RH70 controller on PDP–11/70 systems. All other systems use the RM02 with the RH11 controller.

4.1.4 RP04, RP05, RP06, and RP07 Disks

The RP04 or RP05 (RH11-RH70 controller/RP04 or RP05 disk packs) disk packs consist of 19 data surfaces and a moving read/write head. Both offer large storage capacity with rapid access time. The RP06 disk pack has approximately twice the capacity of the RP04 or RP05. The RP07 fixed-media disk has approximately three times the capacity of the RP06.

4.1.5 RK11/RK05 or RK05F Cartridge Disks

The RK11 controller/RK05 DECpack cartridge disk is an economical storage system for medium-volume, random-access storage. The removable disk cartridge offers the flexibility of large offline capacity with rapid transfers of files between online and offline units without necessitating copying operations. The RK05F has twice the storage capacity of the RK05 and has a fixed (nonremovable) disk cartridge.

4.1.6 RL11/RL01 or RL02 Cartridge Disk

The RL01 is a low-cost, single-head-per-surface disk with a burst data transfer rate of 512-Kb/s. The storage capacity of the RL02 is twice that of the RL01.

4.1.7 RK611/RK06 or RK07 Cartridge Disk

The RK611 controller/RK06 cartridge disk is a removable, random-access, bulk-storage system with three data surfaces. The storage capacity is 6,944,256 words per disk pack. The system, expandable to eight drives, is suitable for medium to large systems.

The RK611 controller/RK07 cartridge disk is generally similar to the RK611/RK06, except storage capacity is increased to approximately 13,905,400 words per disk pack. Both RK06 and RK07 disks can use the same RK611 controller; mixing RK06 and RK07 disks on the same controller is permitted.

4.1.8 RX11/RX01 Flexible Disk

The RX11 controller/RX01 flexible disk is an economical storage system for low-volume, random-access storage. Data is stored in twenty-six 64-word sectors per track; there are 77 tracks per disk. Data may be accessed by physical sector or logical block. If logical or virtual block I/O is selected, the driver reads four physical sectors. These sectors are interleaved to optimize data transfer. The next logical sector that falls on a new track is skewed by six sectors to allow for track-to-track switch time. Physical block I/O provides no interleaving or skewing and provides access to all 2002 sectors on the disk. Logical or virtual I/O starts on track 1 and provides access to 494 logical blocks.

4.1.9 RX211/RX02 Flexible Disk

The RX211 controller/RX02 flexible disk is an economical storage system for low-volume, random-access storage. It is capable of operating in either an industry-standard, single-density mode (as stated for the RX11/RX01 flexible disk), or a double-density mode (not industry standard). In the single-density mode, each drive can store data exactly as stated in Section 4.1.8. In the double-density mode, data is stored in twenty-six 128-word sectors per track; there are 77 tracks per disk. The RX211/RX02 operating in the single-density mode can read disks written by an RX11/RX01 flexible disk system. In addition, disks written by the RX211/RX02 operating in the single-density mode can be read by the RX11/RX01 flexible disk system.

4.1.10 ML-11 Disk Emulator

The ML-11 is a fast, random-access, block-mode MOS memory system. The RSX–11M–PLUS operating system treats the ML-11 as a disk. However, because it is not a disk, the statistics in Table 4–1 do not apply. Unlike a disk, the number of bytes per drive varies. One ML-11 provides from 512 blocks to 8192 blocks of storage.

4.1.11 KDA50, UDA50/RA60/RA80/RA81 Disks

The KDA50 or UDA50 controller is an intelligent disk controller that contains a high-speed microprogrammed processor capable of performing all disk functions, including data handling, error detection and correction, and optimization of disk drive activity and data transfers. The controller optimizes disk activity by reordering QIO$s. Therefore, QIO$ macros may not complete in the order in which they were issued. The types of drives that can be connected to the KDA50 or UDA50 controllers are the RA60 disk drive, which has a removable disk pack, and the RA80, RA81, and RA82, all of which are fixed media drives. (For data capacities and rates, see Table 4–1.) Up to four of these drives can be connected to a KDA/UDA, in any desired combination.

The KDA/UDA controller can perform an extensive self-test on power-up or initialization.

4.1.12 RC25 Disk Subsystem

The RC25 disk subsystem consists of a fixed-media drive and a removable-media drive, both of which revolve on the same spindle and share the same head mechanics. Each drive is a logical unit, so each RC25 disk subsystem consists of two logical units.

The RC25 Subsystem combines, in one package, a controller and a single disk drive that has a removable disk and a fixed disk. These disks reside in the drive as two separate logical units on a single spindle. Their size is the same. Both are single 8-inch disks with two surfaces, and both disks have the same data capacity. But mechanically they are different: One is a removable front-loading cartridge disk, while the other cannot be removed from the drive. The drive contains loadable Winchester heads.

RC25 subsystems are available in two types: a master drive that contains its own controller, and a slave drive, which must be connected to an RC25 master drive. Each RC25 master drive can support one RC25 slave drive. The added-on disk drive is a slave to the disk subsystem that has the controller. A master–slave configuration would contain four logical units.

4.1.13 RD31 Fixed 5.25-Inch Disk

The RD31 disk drive is a 5.25-inch fixed disk with Winchester-type heads. The RD31 is soft sectored and field formattable. The maximum capacity of the RD31 is 20 Mb.

4.1.14 RX33 5.25-Inch Half-Height Disk

The RX33 disk drive is a half-height, 5.25-inch single flexible disk. It operates as a dual speed, double-sided, diskette drive and has a maximum capacity of 1.2 Mb. The RX33 requires the RQDX3 disk controller, supports RX33 formatting, and can perform read/write operations for both RX33 and RX50 diskettes.

4.1.15 RD51 Fixed 5.25 Disk/RX50 Flexible 5.25 Disk

This subsystem consists of a hard disk (RD51) and flexible disk (RX50) combination, and a RQDX1/RQDX2 controller. In combination, they are a mass-storage medium for small systems. The basic configuration for this subsystem is an RD51 fixed-disk drive and an RX50 flexible, dual-disk drive. In this configuration, the RD51 is the system device and the RX50 is a data or a backup device, or both. The RX50 dual disk is addressed as two separate units resulting in a basic configuration of three disk units. Also, you can add another RD51 to increase storage capacity. Some of the characteristics of the RD/RX drives are given in Table 4–1 and in the following paragraphs.

The RD51 disk drive is a 5.25-inch fixed disk with Winchester-type heads. It has two disks with four data surfaces. The RD51 is soft sectored and field formattable. The headers for each sector contain the sector's cylinder number, head number, and sector number. The sector number is the logical sector number (0-15) that reflects the sector interleave of the disk.

The RX50 dual diskette drive is a compact, mass-storage drive with two access slots. Each slot can hold a single-sided 5.25-inch flexible disk. These diskettes are firm sectored and are not field formattable. Every track has sectors numbered from 1 to 10. The two diskettes share the same head transport mechanism.

RSX–11M–PLUS and Micro/RSX also support the RUX50 UNIBUS interface for the RX50 dual diskette drive and the RX180 IBM-compatible diskette drive.

4.1.16 RD52 Fixed 5.25-Inch Disk

The RD52 disk drive is a 5.25-inch fixed disk with Winchester-type heads. The RD52 is soft sectored and field formattable. The maximum capacity of the RD52 is 31 Mb.

4.1.17 RD53 Fixed 5.25-Inch Disk

The RD53 disk drive is a 5.25-inch fixed disk with Winchester-type heads. The RD53 is soft sectored and field formattable. The maximum capacity of the RD53 is 71 Mb.

4.1.18 RD54 Fixed 5.25-Inch Disk

The RD54 disk drive is a 5.25-inch fixed disk with Winchester-type heads. The RD54 is soft sectored and field formattable. The maximum capacity of the RD54 is 159 Mb.

4.2 Get LUN Information Macro

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for disks (a bit setting of 1 indicates that the described characteristic is true for disks):

Bit	Setting	Meaning
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	1	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	1	Mass-storage device
7	X	User-mode diagnostics supported (device dependent)
8	X	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	1	Device mountable as a Files–11 volume
15	1	Device mountable

Words 3 and 4 of the buffer contain the maximum logical block number. Note that the high byte of U.CW2 is undefined. Your task should clear the high byte in the buffer before using the block number. For DU-type disks, these two words are undefined until the device has been mounted at least once. Word 5 indicates the default buffer size, which is 512 bytes for all disks.

4.3 QIO$ Macro

This section summarizes the standard and device-specific QIO functions for disk drivers.

4.3.1 Standard QIO$ Functions

Table 4–2 lists the standard functions of the QIO$ macro that are valid for disks.

Table 4–2: Standard QIO$ Functions for Disks

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Kill I/O[1]
QIO$C IO.RLB,..., <stadd,size,,blkh,blkl>	Read logical block
QIO$C IO.RVB,..., <stadd,size,,blkh,blkl>	Read virtual block
QIO$C IO.WLB,..., <stadd,size,,blkh,blkl>	Write logical block
QIO$C IO.WLC,..., <stadd,size,,blkh,blkl>	Write logical block followed by write-check[2]
QIO$C IO.WVB,..., <stadd,size,,blkh,blkl>	Write virtual block

[1] In-progress disk operations are allowed to complete when IO.KIL is received because they take such a short time. I/O requests that are queued when IO.KIL is received are killed immediately. An IE.ABO status is returned in the I/O status doubleword.

[2] Not supported on RX01 or RX02 flexible disks.

Parameters

stadd

Specifies the starting address of the data buffer (must be on a word boundary).

size

Specifies the data buffer size in bytes (must be even, greater than 0, and, for the RP02 and RP03, also a multiple of 4 bytes).

blkh/blkl

Specifies block high and block low, combining to form a double-precision number that indicates the actual logical/virtual block address on the disk where the transfer starts; blkh represents the high 8 bits of the address, and blkl represents the low 16 bits.

IO.RVB and IO.WVB are associated with file operations (see the *RSX-11M-PLUS and Micro/RSX I/O Operations Reference Manual*). For these functions to be executed, a file must be open on the specified logical unit number (LUN) if the volume associated with the LUN is mounted. Otherwise, the virtual I/O request is converted to a logical I/O request using the specified block numbers.

Note

When writing a new file using QIOs, the task must explicitly issue .EXTND File Control Services (FCS) library routine calls as necessary to reserve enough blocks for the file, or the file must be initially created with enough blocks allocated for the file. In addition, the task must put an appropriate value in the File Descriptor Block (FDB) for the end-of-file block number (F.EFBK) before closing the file. (Refer to the .EXTND routine description in the *RSX-11M-PLUS and Micro/RSX I/O Operations Reference Manual.*)

Each disk driver supports the subfunction bit IQ.X: inhibit retry attempts for error recovery. You use this subfunction bit by using it in a logical OR with the desired QIO; for example:

```
QIO$C IO.WLB!IQ.X,...,<stadd,size,,blkh,blkl>
```

The IQ.X subfunction permits you to specify retry algorithms for applications in which data reliability must be high.

The overlapped seek drivers for RSX-11M-PLUS support subfunction bit IQ.Q, which queues the request immediately without doing a seek (that is, it uses implied seeks).

4.3.2 Device-Specific QIO$ Functions

The device-specific functions of the QIO$ macro are valid for the RX01/RX02/RL01/RL02 disk drives only; they are shown in Table 4-3.

Table 4-3: Device-Specific Functions for the RX01/RX02, RL01/RL02, and RX33 Disk Drives

Format	Function
QIO$C IO.RPB,..., <stadd,size,,,pbn>	Read physical block
QIO$C IO.SEC,...	Sense diskette characteristics (RX02 only)
QIO$C IO.SMD,..., <density,,>	Set media density (RX02 only)
QIO$C IO.WDD,..., <stadd,size,,,pbn>	Write physical block (with deleted data mark) (RX01 and RX02 only)
QIO$C IO.WPB,..., <stadd,size,,,pbn>	Write physical block

Parameters

stadd

Specifies the starting address of the data buffer (must be on a word boundary).

size

Specifies the data buffer size in bytes must be even and greater than 0).

pbn

Specifies the physical block number where the transfer starts (no validation will occur).

density

Specifies the media density as follows:
0 = single (RX01-compatible) density
2 = double density

4.3.3 Device-Specific QIO$ Function for the DUDRV

The DU device driver (DUDRV) supports the device-specific QIO$ function shown in Table 4–4.

Table 4–4: Device-Specific QIO$ Function for the DU: Device Driver

Format	Function
QIO$C IO.RLC,..., <stadd,size,,blkh,blkl>	Read Logical with Read Check modifier

The IO.RLC function is a read logical block followed by a read check. The disk is read twice.

4.4 Status Returns

The error and status conditions listed in Table 4–5 are returned by the disk drivers described in this chapter.

Table 4–5: Disk Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO$ directive was completed successfully. The second word of the I/O status block (IOSB) can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	**I/O request pending** The operation specified in the QIO$ directive has not yet been executed. The IOSB is filled with zeros.
IS.RDD	**Deleted data mark read** A deleted record was encountered while executing an IO.RPB function. The second word of the IOSB can be examined to determine the number of bytes processed (RX01 and RX02 only).
IE.ABO	**Request aborted** An I/O request was queued (not yet acted upon by the driver) when an IO.KIL was issued.
IE.ALN	**File already accessed on LUN** The task attempted to open a file on the physical device unit associated with the specified LUN, but a file has already been opened by the issuing task on that LUN.

Table 4–5 (Cont.): Disk Status Returns

Code	Reason
IE.BLK	**Illegal block number** An invalid logical block number (LBN) was specified. This code would be returned, for example, if block 4800 were specified for an RK05 disk, on which legal block numbers extend from 0 to 4799. IE.BLK would also be returned if an attempt was made to write on the last track of an RK06 disk. (See Section 4.5.)
IE.BBE	**Bad block error** The disk sector (block) being read was marked as a bad block in the header word. Data cannot be written on or read from a bad block.
IE.BYT	**Byte-aligned buffer specified** Byte alignment was specified for a buffer, but only word alignment is legal for disk. Alternatively, the length of a buffer is not an appropriate number of bytes. For example, all RP03 and RP02 disk transfers must be multiples of 4 bytes.
IE.DNR	**Device not ready** The physical device unit specified in the QIO$ directive was not ready to perform the desired I/O operation.
IE.FHE	**Fatal hardware error** The controller is physically unable to reach the location where I/O operation is to be performed. The operation cannot be completed.
IE.IFC	**Illegal function code** A function code was specified in an I/O request that is invalid for disks.
IE.NLN	**File not accessed on LUN** The task attempted to close a file on the physical device unit associated with the specified LUN, but no file was currently open on that LUN.
IE.NOD	**Insufficient buffer space. Caller's nodes exhausted** Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a secondary control block. For example, if a task attempts to open a file, buffer space for the window and file control block must be supplied by the Executive. This code is returned when there is not enough space for this operation.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line.
IE.OVR	**Illegal read overlay request** A read overlay was requested, and the physical device unit specified in the QIO$ directive was not the physical device unit from which the task was installed. The read overlay function can only be executed on the physical device unit from which the task image containing the overlays was installed.

Table 4–5 (Cont.): Disk Status Returns

Code	Reason
IE.PRI	**Privilege violation** The task that issued the request was not privileged to execute that request. For disk, this code is returned if a nonprivileged task attempts to read or write a mounted volume directly (that is, using IO.RLB or IO.WLB). Also, this code is returned if any task attempts to attach a mounted volume.
IE.SPC	**Illegal address space** The buffer specified for a read or write request was partially or totally outside the address space of the issuing task, or a byte count of 0 was specified.
IE.VER	**Parity error on device** After the system's standard number of retries has been attempted upon encountering an error, the operation still could not be completed. For disk, unrecoverable errors are usually parity errors.
IE.WCK	**Write-check error** An error was detected during the write-check portion of an operation.
IE.WLK	**Write-locked device** The task attempted to write on a disk that was write-locked.

4.5 Programming Hints

The following sections describe programming hints you may find helpful for developing drivers for disk devices.

4.5.1 UDA50 QIO$C IO.ATT Before GLUN$

The UDA50 dynamically updates the system database to reflect the characteristics of the UDA50. Therefore, your task should issue a QIO$ IO.ATT function before requesting the device's characteristics with the Get LUN directive.

4.5.2 RX02 QIO$C IO.SEC Before GLUN$

The RX02 driver (DYDRV) dynamically updates the system database to reflect the characteristics of the media in the RX02 drive. Therefore, your task should issue a QIO$C IO.SEC (sense characteristics) function before requesting the device's media characteristics with the GLUN$ directive.

4.5.3 Bad Sector Track on Disks

For the RK611 controller/RK06 or RK07 disk, the RL11 controller/RL01 or RL02 disk, RM02 disk, RM03 disk, RM05 disk, RM80 disk, and RP07 disk, the driver write-protects the last track of the cartridge. This track contains the factory-recorded, bad-sector file.

4.5.4 Stalling Input and Output

Because two RC25 disk units revolve on the same spindle and share the same head mechanics, you must spin down both units of a subsystem in order to spin down one unit. You cannot access either unit until the subsystem is spun up again. Because you must spin down the drive any time you want to insert or remove a disk from the removable-media unit, the device driver (DUDRV) allows you to spin down the subsystem and still retain context on the fixed-media unit, provided it is mounted as a Files–11 or foreign volume. It does this by postponing input and output to the fixed-media unit until the subsystem is spun up again and the heads are reloaded. This is called stalled I/O.

When the driver receives an I/O request that it cannot process because the drive is spun down, it issues the following message to the console:

```
ddnn: - I/O stalled
```

When the drive is spun up again and I/O to the device is resumed, the driver issues the following message to the console:

```
ddnn: - I/O resumed
```

Note that because the only reason you would want to spin down the disk on a running system would be to replace the removable disk, and you would never specifically need to spin down the fixed-media unit, I/O is never stalled to the removable-media unit. The removable-media unit behaves like any other disk on an RSX–11M–PLUS or Micro/RSX system: if you spin it down, context is lost.

Stalling I/O to an RC25 subsystem affects the system's performance. If you initiate an operation requiring I/O to a stalled unit, you will not receive a timely response to the request. Although the I/O request is queued to the device driver, the driver ignores the request until the drive is loaded and the unit is ready. The driver then resumes processing requests. Note, however, that an operation can continue as long as it does not require access to the unit whose I/O is stalled.

Sometimes an operation that does not involve stalled-I/O units is delayed as well. For example, assume that your system disk is in the fixed-media unit and that you spin down a subsystem in order to change the disk pack in the removable-media unit. If a user then initiates an operation requiring a task to be loaded from the fixed unit, the loader issues a queued I/O request to the fixed unit. However, the device driver does not respond to this request immediately, since the subsystem is spun down. Also, because the loader cannot service additional tasks until it loads the current task from the disk, load operations to other disks on the system remain in the loader's work queue until the current load operation completes.

Note

Like the loader, the Files–11 Ancillary Control Processor (Files–11 ACP or F11ACP) is another single-threaded task that may delay response time when I/O is stalled to the RC25. To avoid this delay, you should always install a unique ACP for the RC25 fixed-media units (see the MCR command MOU in the *RSX–11M–PLUS MCR Operations Manual* or the DCL command MOUNT in the *RSX–11M–PLUS Command Language Manual*. Micro/RSX users may also want to refer to the *Micro/RSX User's Guide, Volume 1*).

System users may find it difficult to distinguish between system crashes and system delays due to stalled I/O. Therefore, it is recommended that, before you spin down an RC25 subsystem, you inform all system users of your intentions.

4.5.5 Dismounting the RC25

You dismount a unit on the RC25 in the same way you dismount a unit on other disk devices, by using either the MCR or DCL command DISMOUNT. However, there are restrictions on using the /UNLOAD qualifier to spin down the disk. Since context may be lost on the removable disk if the subsystem is spun down, all spin down requests are ignored for the fixed unit of the RC25. For the removable disk unit, you must be privileged in order to spin down the device while dismounting it. The privileged status of DISMOUNT/UNLOAD is a safety measure to control who is able to spin down the system disk.

If you are a privileged user, DISMOUNT/UNLOAD issues the following message when the command executes properly:

```
Warning -- All units of multiunit drive will spin down <ddnn:>
```

If you are a nonprivileged user, DISMOUNT/UNLOAD refuses your request to spin down a unit and issues the following message:

```
Warning -- Volume will not spin down <ddnn:>
```

Chapter 5
DECtape II Driver

5.1 Introduction to the DECtape II Driver

The DECtape II (TU58) driver supports TU58 system hardware, providing low-cost, block-replaceable mass storage.

5.1.1 TU58 Hardware

Each TU58 DECtape II system consists of one or two TU58 cartridge drives, one tape drive controller, and one DL11-type serial line interface. Each TU58 drive functions as a random-access, block-formatted, mass-storage device. Each tape cartridge is capable of storing 512_{10} blocks of 512_{10} bytes each. Access time averages 10 seconds. All I/O transfers (commands and data) occur by means of the serial line interface at serial transmission rates of 9600 bps. All read and write check operations are performed by the controller hardware that uses a 16-bit checksum. The controller performs up to eight attempts to read a block, as necessary, before aborting the read operation and returning a hard error; however, whenever more than one read attempt is required for a successful read, the driver is notified so that it can report a soft error message to the error logger.

5.1.2 TU58 Driver

The TU58 driver communicates with the TU58 hardware by means of a serial line interface (DL11); no other interface is required. All data and command transfers between the PDP–11 system and the TU58 are done with programmed I/O and interrupt-driven routines; non-processor requests (NPRs) are not supported.

5.2 Get LUN Information Macro

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for the TU58 (a bit setting of 1 indicates that the described characteristic is true for this device):

Bit	Setting	Meaning
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	1	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	1	Mass-storage device
7	1	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	1	Device mountable as a Files–11 volume
15	1	Device mountable

Words 3 and 4 of the buffer are a double-precision number specifying the total number of blocks on the device; this value is 512_{10} blocks. Word 5 indicates the default buffer size, which is 512_{10} bytes.

5.2.1 QIO MACRO

This section summarizes standard and device-specific QIO functions for the TU58.

5.2.2 Standard QIO Functions

Table 5–1 lists the standard QIO system directive functions for the QIO macro that are valid for the TU58.

Table 5–1: Standard QIO Functions for the TU58

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests[1]
QIO$C IO.RLB,..., <stadd,size,,,lbn>	Read logical block
QIO$C IO.WLB,..., <stadd,size,,,lbn>	Write logical block

[1] In-progress operations are allowed to complete when IO.KIL is received. I/O requests that are queued when IO.KIL is received are killed.

Parameters

stadd

Specifies the starting address of the data buffer (must be on a word boundary).

size

Specifies the data buffer size in bytes (must be even and greater than 0).

lbn

Specfies the logical block number on the cartridge tape where the data transfer starts (must be in the range of 0 to 511).

5.2.3 Device-Specific QIO Functions

The device-specific QIO system directive functions that are valid for the TU58 are shown in Table 5–2.

Table 5–2: Device-Specific QIO Functions for the TU58

Format	Function
QIO$C IO.WLC,..., <stadd,size,,,lbn>	Write logical block with check
QIO$C IO.RLC,..., <stadd,size,,,lbn>	Read logical block with check
QIO$C IO.BLS!IQ.UMD,..., <lbn>	Position tape
QIO$C IO.DGN!IQ.UMD,...	Run internal diagnostics

Parameters

stadd
Specifies the starting address of the data buffer (must be on a word boundary).

size
Specifies the data buffer size in bytes (must be even and greater than 0).

lbn
Specifies the logical block number on the cartridge tape where the data transfer starts (must be in the range of 0 to 511).

Additional details for device-specific QIO functions are provided in the following paragraphs.

5.2.3.1 IO.WLC

The IO.WLC function writes the specified data onto the tape cartridge. A checksum verification is then performed by reading the data just written; data is not returned to the task issuing the function. An appropriate status, based on the checksum verification, is returned to the issuing task.

5.2.3.2 IO.RLC

The IO.RLC function reads the tape with an increased threshold in the TU58's data recovery circuit. This is done as a check to ensure data read reliability.

5.2.3.3 IO.BLS

You can use the IO.BLS function for diagnostic purposes to position the tape to the specified logical block number (LBN). If you specify IO.BLS, you must use the IQ.UMD subfunction (see Chapter 1).

5.2.3.4 IO.DGN

You can use the IO.DGN function for diagnostic purposes to execute the TU58's internal (firmware) diagnostics. Appropriate status information is returned to the issuing task by the I/O status block (IOSB). If you specify IO.DGN, you must use the IQ.UMD subfunction (see Chapter 1).

5.3 Status Returns

Table 5–3 lists the error and status conditions that are returned by the TU58 driver.

Table 5–3: TU58 Driver Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the IOSB can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IE.DNR	**Device not ready** The physical device unit specified in the QIO directive was not ready to perform the desired I/O operation.
IE.IFC	**Illegal function code** A function code was specified in an I/O request that is illegal for the TU58.
IE.FHE	**Fatal hardware error**
IE.TMO	**Timeout error** The TU58 failed to respond to a function within the normal time specified by the driver.
IE.VER	**Parity error on device** After the system's standard number of retries (8) has been attempted upon encountering an error, the operation still could not be successfully completed.
IE.WLK	**Cartridge write-locked** The task attempted to write on a tape cartridge that is physically write-locked.

Chapter 6
Magnetic Tape Drivers

6.1 Introduction to the Magnetic Tape Drivers

RSX–11M–PLUS and Micro/RSX support a variety of magnetic tape devices. However, note that Micro/RSX supports only the following magnetic tape devices:

- TSV05
- TK25
- TK50

Table 6–1 summarizes these devices and subsequent sections describe them in greater detail.

Table 6–1: Standard Magnetic Tape Devices

Device Driver	Channels	Recording Density (Frames/ Inch)	Tape Speed (Inches/ Second)	Maximum Data Rate Units	Recording Transfer Method (Bytes/ Second)
TE10 TU10 MTDRV	9 7 or 9	7-channel: 200, 556 or 800 9-channel: 800	45	36,000	NRZI
TE16,TU16 MMDRV	9	800/1600	45	800 bpi: 36,000 1600 bpi: 72,000	NRZI or PE[1]
TU45 MMDRV	9	800/1600	75	800 bpi: 60,000 1600 bpi: 120,000	NRZI or PE[1]
TU77 MMDRV	9	800/1600	125	800 bpi: 100,000 1600 bpi: 200,000	NRZI or PE[1]
TS03 MTDRV	9	800	15	12,000	NRZI
TS11 MSDRV	9	1600	45	72,000	PE[1]
TU80 MSDRV	9	1600	25[2] 100[3]	40,000[2] 160,000[3]	PE[1]

[1] Phase encoded

[2] Low speed

[3] High speed

Table 6–1 (Cont.): Standard Magnetic Tape Devices

Device Driver	Channels	Recording Density (Frames/ Inch)	Tape Speed (Inches/ Second)	Maximum Data Rate Units	Recording Transfer Method (Bytes/ Second)
TU81 MUDRV	9	1600/6250	25[2] 75[3] 25[2] 75[3]	40,000 120,000 156,000 469,000	PE[1] PE[1] GCR GCR
TSV05 MSDRV	9	1600	25	40,000	PE[1]
TK25 MSDRV	s.s.[4]	8000	55	55,000 bit-serial data tracks recorded serial serpentine	Modified GCR
TK50 MUDRV	s.s.[4]	6667	75[5]	45,000 bit-serial data tracks recorded serial serpentine	Modified FM

[1] Phase encoded

[2] Low speed

[3] High speed

[4] Serial serpentine

[5] In streaming mode

6.1.1 TE10/TU10/TS03 Magnetic Tape

The TE10/TU10/TS03 consists of a TM11 controller with a TE10, TU10, or TS03 transport. It is a low-cost, high-performance system for serial storage of large volumes of data and programs in an industry-compatible format. All recording is non-return to zero inverted (NRZI) format.

6.1.2 TE16/TU16/TU45/TU77 Magnetic Tape

The TE16/TU16/TU45/TU77 consists of an RH11/RH70 controller, a TM02 or TM03 formatter, and a TE16/TU16/TU45/TU77 transport. They are quite similar to the TE10/TU10 but are MASSBUS devices, with a common controller, a specialized formatter, and drives. Recording is either 800 bits per inch (bpi) NRZI or 1600 bpi phase encoded (PE).

6.1.3 TS11/TU80 Magnetic Tape

The TS11 and TU80 are integrated subsystems. Each has a drive, a controller, and a formatter. The hardware is microprocessor controlled for all operations, including I/O transfers and tape motion, and it has comprehensive (internal) diagnostic test execution. Recording is 1600 bpi PE.

The TS11 operates in conventional start and stop mode while the TU80 operates at either low speed (start and stop mode) or high speed (streaming mode). Tape speed is microprocessor controlled.

6.1.4 TSV05 Magnetic Tape

The TSV05 tape subsystem is a Q-bus device. It is an integrated subsystem with a drive, a controller, and a formatter. The hardware is microprocessor controlled for all operations, including I/O transfers, tape motion, and it has comprehensive (internal) diagnostic test execution. Recording is 1600 bpi PE. The TSV05 operates at 25 inches per second.

6.1.5 TK25 Magnetic Tape

The TK25 consists of a TKQ25 controller for the Q-bus and a TK25 streaming tape drive. The integrated subsystem consists of a tape drive and controller/formatter. The TK25 uses a DC600A 1/4-inch tape cartridge and stores data on serial data tracks in a serial serpentine recording method. The TK25 has storage capacity of 60 megabytes (Mb) for 8-kilobyte (Kb) data records. Data recording is an 8000 bpi, modified GCR (group cyclical recording) method.

6.1.6 TK50 Magnetic Tape

The TK50 is an integrated subsystem that consists of a controller for the Q-bus (TQK50) or a controller for the UNIBUS (TUK50), and a TK50 streaming tape drive. The controller handles all error recovery and correction, and internally buffers multiple outstanding commands. The tape drive reads and writes data on a 1/2-inch tape cartridge that is recorded at 6667 bpi on serial data tracks in a serial serpentine recording (Modified Frequency Modulation) method. The tape speed is 75 inches per second in streaming mode and the storage capacity is approximately 94 Mb irrespective of record size. There is one drive for each controller.

6.1.7 TU81 Magnetic Tape

The TU81 is a 9-track streaming tape drive that reads and writes data at either 6250 bpi (GCR) or 1600 bpi (PE) on 1/2-inch tape. The TU81 internally buffers multiple outstanding commands. The tape transport speed is 25 or 75 inches per second and is microprocessor controlled. At 6250-bpi density, the drive can store up to 140 Mb on a standard 2400-foot reel. The TU81 has its own UNIBUS controller (one drive per controller).

6.2 Get LUN Information Macro

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for magnetic tapes (a bit setting of 1 indicates that the described characteristic is true for magnetic tapes):

Bit	Setting	Meaning
0	0 or 1	Record-oriented device (0 if the tape is mounted Files–11, 1 if is not mounted or if it is mounted foreign)
1	0	Carriage-control device
2	0	Terminal device
3	0	File-structured device
4	0 or 1	Single-directory device (0 if the tape is not mounted, 1 if it is)
5	1	Sequential device
6	1	Mass-storage device
7	0 or 1	User-mode diagnostics supported[1]
8	0 or 1	MASSBUS device (set only for TE16, TU16, TU45, TU77 drives interfaced by means of an RH70 controller)[1]
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0 or 1	Device mountable as a Files–11 volume[1]
15	0 or 1	Device mountable[1]

[1]System generation and device-dependent characteristic.

Word 3 is used by Digital Equipment Corporation for tape density information. Word 4 of the buffer is undefined; word 5 indicates the default buffer size.

6.3 QIO$ Macro

This section summarizes standard and device-specific QIO$ functions for the magnetic tape drivers.

6.3.1 Standard QIO$ Functions

Table 6–2 lists the standard functions of the QIO$ macro that are valid for magnetic tape.

Table 6–2: Standard QIO$ Functions for Magnetic Tape

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,..., <stadd,size>	Read logical block (read tape into buffer)
QIO$C IO.RVB,..., <stadd,size>	Read virtual block (read tape into buffer)
QIO$C IO.WLB,..., <stadd,size>	Write logical block (write buffer contents to tape)
QIO$C IO.WVB,..., <stadd,size>	Write virtual block (write buffer contents to tape)

Parameters

stadd

Specifies the starting address of the data buffer. It may be on a byte boundary for MSDRV devices. Otherwise, it must be on a word boundary.

size

Specifies the data buffer size in bytes. Size must be even, greater than 0, and, for a write, must be at least 14 bytes. For MSDRV or MUDRV devices, the data transfer size may be an odd or even number of bytes.

6.3.1.1 IO.KIL

IO.KIL causes I/O termination upon the occurrence of:

- A select error (not applicable to TK50)
- Error recovery
- Interrupt servicing
- Driver timeout servicing

Select errors are not issued for MUDRV devices, but any I/O in progress is canceled by IO.KIL.

6.3.2 Device-Specific QIO$ Functions

Table 6–3 lists the device-specific functions of the QIO$ macro that are valid for magnetic tape. Additional details on certain functions appear in the paragraphs that follow.

Table 6–3: Device-Specific QIO$ Functions for Magnetic Tape

Format	Function
QIO$C IO.DSE,...	Data Security Erase (TK50/TU81 only)
QIO$C IO.EOF,...	Write end-of-file (EOF) mark (tape mark)
QIO$C IO.ERS,...	Erase (TE10, TU10, and TK50 are not supported.)
QIO$C IO.RLV,..., <stadd,size>	Read logical block reverse (TE10 and TU10 are not supported.)
QIO$C IO.RWD,...	Rewind unit
QIO$C IO.RWU,...	Rewind and turn unit off line
QIO$C IO.SEC,...	Sense tape characteristics
QIO$C IO.SMO,..., <cb>	Mount tape and set tape characteristics (Unit must be ready with tape at load point.)
QIO$C IO.SPB,..., <nbs>	Space blocks
QIO$C IO.SPF,..., <nes>	Space files
QIO$C IO.STC,..., <cb>	Set tape characteristics

Parameters

stadd

Specifies the starting address of the data buffer. It may be on a byte boundary for MSDRV devices; otherwise, it must be on a word boundary.

size

Specifies the size of the stadd data buffer in bytes. The size must be an even number of bytes greater than 0, and it must be at least 14 bytes for a write. For MSDRV or MUDRV devices, data transfers may be an odd or even number of bytes.

cb

Specifies the characteristic bits to set.

nbs

Specifies the number of blocks to space past (positive if forward, negative if reverse).

nes

Specifies the number of end-of-file (EOF) marks to space past (positive if forward, negative if reverse).

6.3.2.1 IO.RLV

The data appears in the specified buffer in a fashion identical with IO.RLB or IO.RVB, as long as the data block has the same length as the buffer.

6.3.2.2 IO.RWD

Completion of IO.RWD means that the rewind has been initiated, except for MSDRV (MS) devices. For MSDRV devices, completion of IO.RWD indicates that the rewind to beginning-of-tape (BOT) has completed. Additional requests for operations on that controller may then be queued. However, a request for the same unit will be queued by the driver until load point (BOT) is reached.

6.3.2.3 IO.RWU

You normally use IO.RWU when operator intervention is required (for example, to load a new tape). The operator must turn the unit back on line manually before subsequent operations can proceed.

6.3.2.4 IO.ERS

IO.ERS causes an erase of 3 inches of (write blank) tape, effectively providing an extended interrecord gap. (IO.ERS is not supported on TU10 and TE10.)

6.3.2.5 IO.DSE

IO.DSE causes the TK50 and TU81 to erase from the current position to the physical end-of-tape (EOT) and then to rewind the tape to beginning-of-tape (BOT).

6.3.2.6 IO.SEC

IO.SEC causes a return of the tape characteristics in the second I/O status word. The tape characteristic bits are defined as follows:

Bit	Meaning When Set	Can Be Set by IO.SMO and IO.STC
0	For TU10, 556-bpi density (7-channel). Reserved for TE16, TU16, TU45, TU77, TU81, TS11, TK25, and TK50.	X
1	For TU10, 200-bpi density (7-channel). For TS11, TU80, and TSV05, TSU05, TK25, swap byte mode (read/write). Data buffer size should be in even bytes. Reserved for TE16, TU16, TU45, TK50, TU77, and TU81.	X

Bit	Meaning When Set	Can Be Set by IO.SMO and IO.STC
2	For TU10, core-dump mode (7-channel, see below). Reserved for TE16, TU16, TS11, TU45, TU77, TU80, TU81, TSV05, TSU05, TK25, and TK50.	X
3	For TU10, even parity (default is odd). For others, odd parity. (Not selectable for TS11, TK50, TU80, and TU81.)	X
4	Tape is past end-of-tape (EOT).	
5	Last tape command encountered end-of-file (EOF) in a forward tape direction.	
6	Writing is prohibited.	X
7	Writing with extended interrecord gap is prohibited (that is, no recovery is attempted after write error).	X
8	Select error on unit (not on TK50 or TU81).	
9	Unit is rewinding.	
10	Tape is physically write-locked.	
11	For TE10, TU10, TK50 and TS03, reserved. For the TU81, default 6250 bpi. If bit 11 is set, 1600 bpi. For all other tapes, default is 800 bpi. If bit 1 is set, 1600-bpi density.	X
12	For TU10, drive is 7-channel. For all other tapes, reserved.	
13	Tape is at load point (BOT).	
14	Tape is at end-of-volume (EOV).	
15	Tape is past EOV (reserved for driver; always 0 when read by your task).	

In core-dump mode (TU10 only, 800-bpi density, and 7-channel), each 8-bit byte is written on two tape frames, 4 bits per frame. In other modes on 7-channel tape, only 6 low-order bits per byte are written.

For the TS11/TU80/TSV05/TSU05 1600-bpi density is always selected (bit 11=1). Bit 11 cannot be modified by either the IO.SMO or IO.STC functions. For drives that use the TM03 controller, this bit can be either set or cleared; however, once the tape is moved from the load BOT position, the device driver modifies this bit to reflect the actual density of the tape currently mounted. You cannot change bit 11 once the tape is moved beyond BOT.

6.3.2.7 IO.SMO

Use the IO.SMO function as a combination of the sense (IO.SEC) and set (IO.STC) tape characteristics functions. Unlike IO.STC, however, the IO.SMO function requires that the unit be ready and the tape be at load point (BOT). If either of these conditions is not met, the function returns an error status code of IE.FHE (refer to Table 6–4).

You should use the IO.SMO function to set the characteristics of a newly loaded tape. If the IE.FHE error code is returned, the tape drive is not on line and is not at BOT.

6.4 Status Returns

The error and status conditions listed in Table 6–4 are returned by the magnetic tape drivers described in this chapter.

Table 6–4: Magnetic Tape Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO$ directive was completed successfully. The second word of the I/O status block (IOSB) can be examined to determine the number of bytes processed, if the operation involved reading or writing. This code is also returned if nbs equals 0 in an IO.SPB function or if nes equals 0 in an IO.SPF function.
IS.PND	**I/O request pending** The operation specified in the QIO$ directive has not yet been completed. The IOSB is filled with zeros.
IE.ABO	**Operation aborted** The specified I/O operation was canceled by IO.KIL while in progress or while still in the I/O queue.
IE.BBE	**Bad block** A bad block was encountered while reading or writing and the error persisted after nine retries. For TM11, IE.BBE may also indicate that a bad tape error (BTE) has been encountered. The status return IE.BBE does not apply to MSDRV or MUDRV devices.
IE.BYT	**Byte-aligned buffer specified** Byte alignment was specified for a buffer, while only word alignment is legal for the QIO. Alternatively, the length of a buffer is not an even number of bytes.
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, and not that the unit was attached by another task.
IE.DAO	**Data overrun** On a read, a record exceeded the stated buffer size. The final portion of the buffer is checked for parity but is not transferred into memory.

Table 6–4 (Cont.): Magnetic Tape Status Returns

Code	Reason
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.DNR	**Device not ready** The physical device unit specified in the QIO$ directive was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: • A timeout occurred on the physical device unit (that is, an interrupt was lost). • A vacuum failure occurred on the magnetic tape drive. • While trying to read or space, the driver detected blank tape. • The LOAD switch on the physical drive was switched to the off position. • The unit failed internal diagnostic tests (TS04 only).
IE.EOF	**End-of-file encountered** An EOF (tapemark) was encountered.
IE.EOT	**End-of-tape encountered** The EOT (physical end-of-volume) was encountered while the tape was moving in the forward direction for a write or a write tape mark operation. The IE.EOT code is returned continually in the IOSB until the EOT marker is passed in the reverse direction. IE.EOT is not returned on a read operation. A 10-foot length of tape extends past the EOT marker, which is useful for writing data and markers, such as volume trailer labels. The physical EOT for MUDRV (MU) devices is defined as the end of usable recorded area, which is located in the tape trailer area. This area begins at the EOT marker and extends through a length that depends on the tape format and is controller dependent.
IE.EOV	**End-of-volume encountered (unlabeled tape)** On a forward space function, the logical end-of-volume (EOV) was encountered. An end-of-volume (EOV) is two consecutive end-of-file marks (EOF) or a beginning-of-tape mark (BOT) followed by an EOF. The tape is normally left positioned between the two marks.
IE.FHE	**Fatal hardware error** Nonrecoverable hardware error; for example, the magnetic tape unit is not ready or the tape is not at load point, or both, when IO.SMO is issued.
IE.IFC	**Illegal function code** An invalid function (or subfunction bit) was specified in a magnetic tape I/O request. Refer also to Section 6.4.3.

Table 6–4 (Cont.): Magnetic Tape Status Returns

Code	Reason
IE.OFL	**Device off line** The physical device unit associated with the logical unit number (LUN) specified in the QIO$ directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	**Illegal address space** The buffer specified for a read or write request was partially or totally outside the address space of the issuing task. For magnetic tape, this code is also returned if a byte count of 0 was specified or if your task attempted to write a block that was less than 14 bytes long.
IE.VER	**Parity error on device** After the system's standard number of retries had been attempted upon encountering an error, the operation still could not be completed. For magnetic tape, this code is returned in the case of cyclic redundancy check (CRC) or checksum errors or when a tape block could not be read.
IE.WLK	**Write-locked device** The task attempted to write on a magnetic tape unit that was physically write-locked. Alternatively, tape characteristic bit 6 was set by the software to write-lock the unit logically.

After processing a QIO$ request, the magnetic tape driver returns two status words. The first word contains one of the I/O status codes listed in Table 6–4.

For successful QIO$ execution (IS.SUC) or read requests (IE.DAO), the second I/O status word may contain further information. The operations for which this is true, and the information returned, are shown in Table 6–5. For all other cases this word is undefined.

Table 6–5: Information Contained in the Second I/O Status Word

I/O Function Code	Information Returned	
	IS.SUC	IE.DAO
IO.RLB	Number of bytes transferred	Number of bytes in tape record[1]
IO.RLV	Number of bytes transferred	Number of bytes in tape record[1]
IO.RVB	Number of bytes transferred	Number of bytes in tape record[1]
IO.SEC	Tape characteristics word	
IO.SPB	Number of records spaced over	
IO.SPF	Number of files spaced over	
IO.WLB	Number of bytes transferred	
IO.WVB	Number of bytes transferred	

[1]Does not apply to MS devices.

6.4.1 Select Recovery

If a request fails because the desired unit is off line, because no drive has the desired unit number, or because the drive has its power off, the following message is output on the operator's console:

```
*** MTn: -- SELECT ERROR
```

or

```
*** MSn: -- SELECT ERROR
```

The n argument is the unit number of the specified drive.

The driver checks the unit for readiness and repeats the message every 15 seconds until the requesting task is aborted or until the unit is made available. In the latter case, the driver then proceeds with the request.

MUDRV devices (TK50) do not issue select errors. If the drive is taken off line, the condition is treated as tape position lost. The cartridge must be unloaded and loaded in order to access the tape again.

Warning

This action results in the tape unit rewinding to BOT. No recovery by the application is possible in such an event. If the tape was mounted as an American National Standards Institute (ANSI) tape, the tape context maintained by MTAACP is invalid. The tape must be dismounted and remounted in order to reinitialize the data structures used by MTAACP. If the tape was being accessed in write mode, the file being written is incomplete and the tape may no longer be in valid ANSI format.

6.4.2 Retry Procedures for Reads and Writes

If an error occurs during a read (for example, vertical parity error), the recovery procedure depends on the type of magnetic tape in use. Read errors for an MT or MM device are retried by backspacing one record and then by rereading the record in question. If the error persists after nine retries, IE.VER is returned.

Read errors for the MSDRV (MS) devices are retried by rereading the block in error a predetermined number of times. For MS devices, except for TK25, on every eighth reread the block is passed by the tape cleaner blade. If the error persists after a predetermined number of retries, IE.VER is returned.

For MUDRV devices (TK50/TU81), the controller handles error correction and recovery. Except for MU devices, write recovery is the same for all devices. When a write operation fails, the driver attempts the following error recovery procedure:

1. Re-positions the tape
2. Erases 3 inches of tape (resulting in an extended interrecord gap)
3. Retries the write operation

If the error persists after a predetermined number of retries, IE.VER is returned. The requesting task can use IO.STC to prohibit writing with an extended interrecord gap. In this case, the tape is backspaced and the write is retried.

6.4.3 Powerfail Recovery for Magnetic Tapes

If a power failure or loss of vacuum, or both, occurs on a magnetic tape drive, tape position is lost. (Note that an initial system boot simulates a recovery from a power failure.) Additionally, on autoload drives, the tape is positioned at BOT when the unit is turned on line.

To prevent accidental destruction of data currently on tape, the driver maintains a powerfail status indicator. When this indicator is set, the driver disallows any data transfer or tape motion commands until a rewind (IO.RWD), rewind unload (IO.RWU), or mount and set characteristics (IO.SMO) function is issued. These functions clear the powerfail indicator and allow all tape functions to be issued. It is also possible to issue the set and sense characteristics functions (IO.STC and IO.SEC) while the powerfail indicator is set. These functions, however, do not clear the bit.

All functions other than those just described are considered invalid and cause the return of the IE.IFC (invalid function) error code to the requesting task. In situations where a tape is currently a mounted volume, the tape should be dismounted and then remounted before use. In doing this, the rewind command is issued, thereby clearing the powerfail indicator.

6.5 Programming Hints

This section contains important information about programming the magnetic tape drivers described in this chapter.

6.5.1 Issue Powerfail QIOs for TM11 Before GLUN$

The TM11A/B device driver dynamically updates the system database to reflect the density characteristics of the TE10/TU10. You should issue the QIO$ functions valid for powerfail before requesting the device's density characteristics with the GLUN$ directive.

6.5.2 Block Size

Each block must contain an even number of bytes: at least 14 for a write and at most 65,534. However, tape usage is more efficient with a larger buffer.

6.5.3 Importance of Resetting Tape Characteristics

A task that uses magnetic tape should always set the tape characteristics to the proper value before beginning I/O operations. The task cannot be certain in what state a previous task left these characteristics. It is also possible that an operator might have changed the magnetic tape unit selection. If the selection switch is changed, the new physical device unit may not correspond to the characteristics of the unit described by the respective Unit Control Block (UCB).

6.5.4 Aborting a Task

If you abort a task while it waits for a magnetic tape unit to be selected, the magnetic tape driver recognizes the abort request within 1 second.

If you abort a task while it waits for a magnetic tape unit to complete a space operation, the magnetic tape driver may allow spacing to the next tape mark.

For the TK50, if you abort a task while it waits for a magnetic tape unit to complete a space operation, the driver may have spaced over some or all of the requested number.

6.5.5 Writing an Even-Parity Zero–NRZI

If an even-parity zero was written normally, it would appear to the drive as blank tape. It is therefore converted. If this conversion is undesirable, you must ensure that no even-parity zeros are output on the tape.

6.5.6 Density Selection

The TM03 controller imposes the following density selection restriction: You cannot mix recording densities on any volume associated with the controller.

Density for write operations is selected when the tape is at the load (BOT) position. Hardware selects the density for read operations during the first read (away from BOT); after the first read, you can determine (sense) tape density by using the IO.SEC function.

6.5.7 End-of-Volume Status (Unlabeled Tape)

The magnetic tape driver detects end-of-volume (EOV) when it spaces over the second of two consecutive tape marks. The tape is left positioned between the two tape marks.

The magnetic tape driver returns the IE.EOV status code only on space operations. IE.EOV is never returned by read operations.

For the purpose of checking for EOV, the driver treats beginning-of-tape (BOT) as a tape mark. Therefore, any forward space operation from BOT that immediately encounters a tape mark returns IE.EOV.

If a space operation stops between two tape marks but does not space over the second one, the driver returns end-of-file (EOF) rather than EOV. Any subsequent space operation from this point that immediately spaces over the second tape mark returns EOV. During IO.SPF operations, the driver considers all tape marks to be files except for BOT and for the second tape mark spaced over at the EOV.

Note that both IO.SPF and IO.SPB operations leave the tape positioned after the tape mark in the direction of travel.

If you want to treat two consecutive tape marks as EOV on read operations, your application must keep track of the tape marks. The magnetic tape driver does not support two consecutive tape marks as EOV on read operations.

6.5.8 Resetting Tape Transport Status or Volume Check

For an MS device, if the tape transport status changes (goes on line or off line), further I/O operations are inhibited. A deliberate I/O sequencing must occur to reset the hardware volume check (VCK) indicator and to allow physical I/O to proceed. This sequencing is accomplished by issuing a sucessful IO.RWD or IO.SMO QIO$ or by including /RW or /REW switches to command requests (such as DMP).

Similarly, for a TK50 or TU81, if the tape transport status changes (goes on line or off line), further I/O operations are inhibited. A deliberate I/O sequencing must occur to allow physical I/O to proceed. This sequencing is done by issuing a successful IO.RWD or IO.SMO QIO$ or by including /RW or /REW switches to command requests (such as DMP).

6.5.9 Issuing QIO$s

Users issuing QIO$s directly to MSDRV/MUDRV must be aware of the following:

- Completion of an IO.RWD request occurs when the MS device reaches BOT.
- Completion of an IO.RWD request occurs when the MU device starts the rewind.
- When the MS or MU device changes status from offline to online or conversely, the MS or MU device inhibits further physical I/O operations. After such a change, the user must issue IO.RWD or IO.SMO requests that succeed before I/O can proceed.
- For the MS or MU device, read/write data transfer features include the following:
  - The data buffer starting address must be on a word boundary.
  - The data transfer size may be an odd or even byte count. The minimum must be 14 bytes.
  - For the MSDRV, you can swap odd and even data bytes by using the tape characteristic bit 1 of IO.SMO or IO.STC requests. When bit 1 is set to 0, no byte swap occurs; when it is set to 1, byte swap does occur. If you use byte swapping, it is recommended that the data buffer size be an even byte count.
- For MU devices, issuing an IO.KIL terminates the "in-progress" I/O operations in reverse order.

Caution

The MU device handles QIO$ requests in a different manner than other devices do. Multiple requests are queued in the controller itself; therefore, the physical end-of-tape (EOT) may be reached before all requests are processed. Thus, with multiple QIO$s it is possible to pull tape off the supply reel.

It is recommended that QIOW$ be used, or that the total size of queued records to be written is not longer than the ANSI standard for the tape trailer size.

The physical end-of-tape (EOT) for MUDRV (MU) devices is defined as the end of usable recorded area, which is located in the tape trailer area. This area begins at the EOT marker and extends through a length that depends on the tape format.

6.6 Block Size on Tapes Mounted /NOLABEL

Under certain conditions, if a file is written to a tape, its block size will be even and one more than the value specified in the DCL command MOUNT. The conditions where this occurs are as follows:

- The tape is mounted with the /NOLABEL qualifier specified.
- The MOUNT command specifies an odd record size.
- The MOUNT command specifies an odd block size.

File Control Services (FCS) adds the padding character, an octal 136 (^) circumflex, to odd-sized blocks due to a hardware restriction; some tape drives will not allow an odd number of bytes to be transferred to or from tape. Therefore, blocks of data are padded with the circumflex character so that blocks of data can be written to tape on any tape drive.

Chapter 7
Line Printer Driver

7.1 Introduction to the Line Printer Driver

The RSX–11M–PLUS and Micro/RSX line printer driver supports the line printers summarized in Table 7–1. Subsequent sections of this chapter describe these printers in greater detail.

Table 7–1: Standard Line Printer Devices

Controller	Printer	Column Width	Character Set	Lines per Minute
KMC11-A Auxiliary Processor				
LP11-C	LP14-C	132	64	890
LP11-D	LP14-D	132	96	650
LP11-F	LP01-F	80	64	170–1110
LP11-H	LP01-H	80	96	170–1110
LP11-J	LP02-J	132	64	170–1110
LP11-K	LP02-K	132	96	170–1110
LP11-R	LP04-R	132	64	1110
LP11-S	LP04-S	132	96	1110
LP11-V	LP05-V	132	64	300
LP11-W	LP05-W	132	96	300
LP11-Y	LP06-Y	132	64	600
LP11-Z	LP06-Z	132	96	460
LP11-GA	LP07	132	96	1200
LP11-EA	LP26	132	64	600

Table 7–1 (Cont.): Standard Line Printer Devices

Controller	Printer	Column Width	Character Set	Lines per Minute
LP11-EB	LP26	132	64/96	600/420
LP11-UA	LP27	132	64/96	1200/800
LS11	LS11	132	62	60-200
LV11	LV01	132	96	500
LA180	LA180	132	96	150
LN01	LN01	Variable	–[1]	600
LN03	LN03	Variable	–[1]	600

[1] Software-selectable fonts not supported by RSX.

7.1.1 KMC-11 Auxiliary Processor

The KMC-11 controller is a microcode-controlled printer controller that supports up to eight line printers. Multiple KMC-11 controllers are allowed. The KMC-11 provides higher performance printing than other controllers and, at the same time, uses fewer central processing unit (CPU) resources. The use of the KMC-11 controller is a system generation option.

7.1.2 LP11 Line Printer

The LP11 is a high-speed line printer available in a variety of models. The LP11 model line consists of band line printers and drum line printers. The drum line printers are impact printers that use one hammer for each column and a revolving drum with uppercase and optional lowercase characters. The LP11-R and LP11-S are fully buffered models that operate at a standard speed of 1110 lines for each minute. The other LP11 drum models have 20-character print buffers. These printers are able to print at full speed if the printed line is no longer than 20 characters. Lines that exceed this maximum are printed at a slower rate. You may use forms with up to six parts. The band line printers are impact printers that have a flat steel belt with raised metal characters on the face. The LP07, LP26, and LP27 offer speeds from 420 to 1200 lines per minute.

7.1.3 LS11 Line Printer

The LS11 is a medium-speed line printer. It has a 20-character print buffer, and lines of 20 characters or less are printed at a rate of 200 lines for each minute. Longer lines are printed at a slower rate.

7.1.4 LV11 Line Printer

The LV11 is a fully buffered, electrostatic printer/plotter that operates at a standard rate of 500 lines for each minute.

7.1.5 LA180 DECprinter

The LA180 is a 180-character-per-second, dot-matrix impact printer. It accepts multipart forms and pages of various lengths and widths.

7.1.6 LN01 Laser Printer

The LN01 is a nonimpact page printer that uses laser imaging combined with xerographic printing. This technology provides letter quality printing at line printer speeds with no noise. Printing is done on standard 8-1/2 inch by 11-inch paper at 12 pages for each minute, which equates to 600 lines for each minute. Contributing to the high print quality is a printer resolution of 300 by 300 dots for each inch. The LN01 offers the speed of a line printer with the advantages of a phototypesetting device.

7.2 Get LUN Information Macro

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for line printers (a bit setting of 1 indicates that the described characteristic is true for line printers):

Bit	Setting	Meaning
0	1	Record-oriented device
1	1	Carriage-control device
2	0	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Mass-storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a Files–11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default size for the device and for line printers the word indicates the width of the printer carriage (that is, 80 or 132 columns).

7.3 QIO$ Macro

Table 7–2 lists the standard functions of the QIO macro that are valid for line printers.

Table 7–2: Standard QIO Functions for Line Printers

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.WLB,..., <stadd,size,vfc>	Write logical block (print buffer contents)
QIO$C IO.WVB,..., <stadd,size,vfc>	Write virtual block (print buffer contents)

Parameters

stadd
Specifies the starting address of the data buffer (may be on a byte boundary).

size
Specifies the data buffer size in bytes (must be greater than 0).

vfc
Specifies a vertical format control character from Table 7–4.

IO.KIL does not cancel an in-progress request unless the line printer is in an offline condition because of a power failure or a paper jam, or because it is out of paper.

The line printer driver supports no device-specific functions.

7.4 Status Returns

Table 7–3 lists the error and status conditions that are returned by the line printer driver described in this chapter.

Table 7–3: Line Printer Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the I/O status block (IOSB) can be examined to determine the number of bytes processed, if the operation involved writing.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not yet been executed. The IOSB is filled with zeros.

Table 7–3 (Cont.): Line Printer Status Returns

Code	Reason
IE.ABO	**Operation aborted** The specified I/O operation was canceled while in progress or while in the I/O queue.
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, and not that the unit was attached by another task.
IE.DNA	**Device not attached** The physical device unit specified an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.IFC	**Illegal function code** A function code was specified in an I/O request that is invalid for line printers.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	**Illegal address space** The buffer specified for a write request was partially or totally outside the address space of the issuing task. Alternatively, a byte count of 0 was specified.

7.4.1 Ready Recovery

The driver determines that the line printer is off line if any of the following conditions occur:

- Paper jam
- Printer out of paper
- Printer turned off line
- Power failure

If the line printer is off line, the following message is output on the operator's console:

```
***LPn: -- NOT READY
```

The argument n is the unit number of the line printer that is not ready.

The driver retries the function that encountered the error condition from the beginning, once every second. It displays the message after m seconds. The value m is defined at system generation to be a value less than 256. The default is 15. The messages occur until you make the line printer ready. If a power failure occurs while printing a line, the entire line is reprinted from the beginning when power is restored.

7.5 Vertical Format Control

Table 7–4 summarizes the meaning of all characters that you can use for vertical format control on the line printer. Any one of these characters can be specified as the vfc parameter in an IO.WLB or IO.WVB function.

Table 7–4: Vertical Format Control Characters

Octal Value	Character	Meaning
040	Blank	**Single space** Output a line feed, print the contents of the buffer, and output a carriage return. Normally, printing immediately follows the previously printed line.
060	Zero	**Double space** Output two line feeds, print the contents of the buffer, and output a carriage return. Normally, the buffer contents are printed two lines below the previously printed line.
061	One	**Page eject** Output a form feed, print the contents of the buffer, and output a carriage return. Normally, the contents of the buffer are printed on the first line of the next page.
053	Plus	**Overprint** Print the contents of the buffer and perform a carriage return, normally overprinting the previous line.
044	Dollar sign	**Prompting output** Output a line feed and then print the contents of the buffer.
000	Null	**Internal vertical format** The buffer contents are printed without addition of vertical format control characters. In this mode, more than one line of guaranteed contiguous output can be printed for each I/O request.

All other vertical format control characters are interpreted as blanks (040_8).

7.6 Programming Hints

This section contains important information about programming the line printer driver described in this chapter.

7.6.1 RUBOUT Character

The line printer driver discards the American Standard Code for Information Interchange (ASCII) character code 177 during output, because a RUBOUT on the LS11 printer causes a RUBOUT of the hardware print buffer.

7.6.2 Print Line Truncation

If the number of characters to be printed exceeds the width of the print carriage, the driver discards excess characters until it receives one that instructs it to empty the buffer and return to horizontal position 1. You can determine if truncation can occur by issuing a Get LUN Information system directive and by examining word 5 of the information buffer. This word contains the width of the print carriage in bytes.

7.6.3 Aborting a Task

If a task is aborted while waiting for the line printer to be readied, the line printer driver recognizes this fact within 1 second.

Chapter 8
Null Device Driver

8.1 Introduction to the Null Device Driver

RSX–11M–PLUS and Micro/RSX provide a driver for a software construct called the "null device." The mnemonic for the null device is NL, and its characteristics are as follows:

- A read from NL returns an end-of-file (EOF) error (IE.EOF).
- A write to NL immediately returns success (IS.SUC).

8.2 Null Device Function

The null device functions as a "black hole" to which your task can direct output, and from which the data so directed never returns. It is particularly useful when you use it with an indirect command file and the MCR command ASN as in the example that follows.

The following shows the contents of a Task Builder (TKB) command file called TESTBLD.CMD:

```
OU:TEST,MP:TEST=IN:[200,220]TEST
/
ASG=TI:2
//
```

This command file uses symbolic device names for the output file, map file, and input file. These names may be reassigned at task-build time. In particular, the following example assigns the map file to the null device and thus the map file is discarded:

```
>ASN SY:=OU: RET
>ASN NL:=MP: RET
>ASN DK1:=IN: RET
>TKB @TESTBLD RET
```

Part II: RSX–11M–PLUS Drivers

Chapter 9
Card Reader Driver

9.1 Introduction to the Card Reader Driver

The RSX–11M–PLUS card reader driver supports the CR11 card reader. This reader is a virtually jam-proof device that reads Electronic Industries Association (EIA) standard 80-column punched cards at the rate of 300 per minute. The hopper can hold 600 cards. This device uses a vacuum picker that provides extreme tolerance to damaged cards and makes card wear insignificant. Cards are riffled in the hopper to prevent sticking. The reader uses a strong vacuum to deliver the bottom card. Because it has a very short card track, only one card is in motion at a time.

9.2 Get LUN Information Macro

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for card readers (a bit setting of 1 indicates that the described characteristic is true for card readers):

Bit	Setting	Meaning
0	1	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	0	Sequential device
6	0	Mass-storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing

Bit	Setting	Meaning
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a Files–11 volume
15	0	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default buffer size, which is 80 bytes for the card reader.

9.3 QIO$ Macro

This section summarizes standard and device-specific QIO functions for the card reader driver.

9.3.1 Standard QIO Functions

Table 9–1 lists the standard functions of the QIO macro that are valid for the card reader.

Table 9–1: Standard QIO Functions for the Card Reader

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests
QIO$C IO.RLB,..., <stadd,size>	Read logical block (alphanumeric)
QIO$C IO.RVB,..., <stadd,size>	Read virtual block (alphanumeric)

Parameters

stadd

Specifies the starting address of the data buffer (may be on a byte boundary).

size

Specifies the data buffer size in bytes (must be greater than 0).

IO.KIL does not cancel an in-progress request unless the card reader is in an offline condition because of a pick, read, stack, or hopper check, because of power failure, or because the RESET button has not been depressed.

9.3.2 Device-Specific QIO Functions

The device-specific functions of the QIO macro that are valid for the card reader are shown in Table 9–2.

Table 9–2: Device-Specific QIO Function for the Card Reader

Format	Function
QIO$C IO.ATA,..., <ast, addr>	Attach for unsolicited card AST
QIO$C IO.RDB,..., <stadd,size>	Read logical block (binary)

Parameters

ast addr

Specifies the address of the asynchronous system trap (AST) processing routine for the function.

stadd

Specifies the starting address of the data buffer (may be on a byte boundary).

size

Specifies the data buffer size in bytes (must be greater than 0).

9.4 Status Returns

A wide variety of error conditions and recovery procedures relate to the use of the card reader. This section describes the following three major ways in which the system reports error conditions:

1. Lights and indicators on the card reader panel are turned on or off to indicate particular operational problems such as read, pick, stack, or hopper checks. Switches are available to turn the reader power on and off and to allow you to reset the error condition after correcting it.
2. A message is output on the operator's console if operational checks or power problems occur.
3. An I/O completion code is returned in the low-order byte of the first word of the I/O status block (IOSB) specified in the QIO macro to indicate success or failure on completion of an I/O function.

The following subsections describe each of these returns in detail.

9.4.1 Card Input Errors and Recovery

Table 9–3 describes all external lights and switches on the reader that indicate to you that a hardware problem has occurred and must be corrected. There are two classes of hardware errors:

- Those requiring you to ready the reader and try the operation again
- Those requiring you to remove the last card from the output stacker, to replace it in the input hopper, and to try the operation again

In the first case, the card reader was unable to read the current card. In the second, the card was read incorrectly and had to be physically removed from the output stacker. The card reader driver restarts a read operation within 1 second after the cards have been replaced in the input hopper.

Table 9–3 summarizes the functions of lights and indicators on the front panel of the card reader. It discusses common operational errors that might be encountered while reading cards and recovery procedures associated with these error conditions.

Table 9–3: Card Reader Switches and Indicators

Indicator	Description
POWER switch	Pushbutton indicator switch (alternate action: pressed for both ON and OFF) **Action:** Controls application of all power to the card reader. When indicator is off, depressing switch applies power to the reader and causes associated indicator to light. When indicator is lit, depressing switch removes all power from the reader and causes indicator to go out. **Recovery:** Card may have been read incorrectly. If possible, restore power by depressing the POWER switch; insert the card again as the first card in the input hopper; and press the RESET switch. In some cases, it may be necessary to restart the program.
READ CHECK indicator	White light **Action:** When lit, indicates that the card just read may be torn on the leading or trailing edges, or it may indicate that the card may have punches in the column positions 0 or 81. Because READ CHECK indicates an error condition, whenever this indicator is lit, it causes the card reader to stop operation and extinguishes the RESET indicator. **Recovery:** Card was read incorrectly. Duplicate the card if necessary, insert the card again as the first card in the input hopper, and press the RESET switch.
PICK CHECK indicator	White light **Action:** When lit, indicates the card reader failed to move a card into the read station after it received a READ command from the controller. Stops card reader operation and extinguishes the RESET indicator. **Recovery:** Card could not be read. Press the RESET switch to try again, or remove the cards from the input hopper smooth the leading edges, replace, and then press the RESET switch.

Table 9–3 (Cont.): Card Reader Switches and Indicators

Indicator	Description
STACK CHECK indicator	White light **Action:** When lit, indicates that the previous card was not properly seated in the output stacker and therefore may be mutilated. Stops card reader operation and extinguishes RESET indicator. **Recovery:** Card may have been read incorrectly and is not positioned properly in the output stacker. Duplicate the card if it is damaged insert the card again as the first card in the input hopper, and press the RESET switch.
HOPPER CHECK indicator	White light **Action:** When lit, indicates that either the input hopper is empty or that the output stacker is full. **Recovery:** Card may have been read incorrectly. Empty the stacker or fill the hopper, insert the card again as the first card in the hopper, and press the RESET switch.
STOP switch	Momentary pushbutton/indicator switch (red light) **Action:** When depressed, immediately lights and drops the READY line, thereby extinguishing the RESET indicator. Card reader operation then stops as soon as the card currently in the read station has been read. The switch has no effect on the system power; it only stops the current operation.
RESET switch	Momentary pushbutton/indicator switch (green light) **Action:** When depressed and released, clears all error flip-flops and initializes card reader logic. Associated RESET indicator lights to indicate that the READY signal is applied to the controller. The RESET indicator goes out whenever the STOP switch is depressed or whenever an error indicator lights (READ CHECK, PICK CHECK, STACK CHECK, or HOPPER CHECK).

9.4.2 Ready and Card Reader Check Recovery

The card reader driver determines that the card reader is not ready if any of the following conditions occur:

- Power failure
- Reset switch not pressed (reader off line)
- Timing error (Two columns were read before the card reader driver input the first column from the card reader.)

The following message is then output on the operator's console:

```
*** CRn: -- NOT READY
```

When a timing error occurs, the operator can proceed with normal card reader operation by performing the following steps:

1. Placing the card reader off line by pressing the STOP button
2. Removing the last card read and inserting it where it is read as the next card
3. Placing the card reader on line by pressing the RESET button

The driver determines that a card reader check has occurred if any of the following conditions occurs:

- Pick error (PICK CHECK)
- Read error (READ CHECK)
- Output stacker error (STACK CHECK)
- Input hopper out of cards (HOPPER CHECK)
- Output stacker full (HOPPER CHECK)

The following message is then output on the operator's console:

```
*** CRn: -- READ FAILURE. CHECK HARDWARE STATUS
```

The n is the unit number of the card reader that is not ready. The operator should correct the error and press the RESET button: The driver attempts the function from the beginning, once every second. The driver displays the message once every m seconds (m is defined at system generation as a value less than 256; the default is 15) until the card reader is readied. In all cases, except pick error, the last card read should be reinserted in the input hopper, as described in Section 9.4.1.

9.4.3 I/O Status Conditions

The error and status conditions listed in Table 9–4 are returned by the card reader driver described in this chapter.

Table 9–4: Card Reader Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully. The second word of the IOSB can be examined to determine the number of bytes processed if the operation involved reading.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not yet been executed. The IOSB is filled with zeros.
IE.ABO	**Operation aborted** The specified I/O operation was canceled while in progress or while still in the I/O queue.
IE.DAA	**Device already attached** The physical device unit specified in an IO.ATT function was already attached by the issuing task.
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.

Table 9–4 (Cont.): Card Reader Status Returns

Code	Reason
IE.EOF	**End-of-file encountered** An end-of-file (EOF) control card was recognized.
IE.IFC	**Illegal function code** A function code was specified in an I/O request that is illegal for card readers.
IE.NOD	**Buffer allocation failure** Dynamic storage space has been depleted, and there was insufficient buffer space available to allocate a card buffer (that is, cards are read into a driver buffer, translated, and then moved to your task's buffer).
IE.OFL	**Device off line** The physical device unit associated with the logical unit number (LUN) specified in the QIO directive was not on line. When the system was booted, a device check indicated that this physical device unit was not in the configuration.
IE.SPC	**Illegal address space** The buffer specified for a read request was partially or totally outside the address space of the issuing task. Alternatively, a byte count of 0 was specified.

9.5 Functional Capabilities

The card reader driver can perform the following functions:

- Read cards in DEC026 format and translate to American Standard Code for Information Interchange (ASCII)
- Read cards in DEC029 format and translate to ASCII
- Read cards in binary format

If the QIO macro specifies the IO.RLB or IO.RVB function, the driver interprets all data as alphanumeric (026 or 029 format). As explained in the next section, control characters indicate whether 026 or 029 is desired. If the QIO macro specifies IO.RDB, the driver interprets all data, including 026 and 029 control characters, as binary.

9.5.1 Control Characters

Table 9–5 lists the multipunched cards that the card reader driver recognizes as control characters. They are never transferred to the buffer of your task or included in the count of transferred bytes in alphanumeric mode. In binary mode, the only control card recognized is binary EOF.

DEC026 is the default translation mode when the system is bootstrapped. This mode remains in effect until explicitly changed by a control card indicating that DEC029 cards follow. After encountering a DEC029 control card, the driver translates all cards in DEC029 format unless another DEC026 control card is encountered. This card overrides the 029 mode specification and indicates that subsequent cards are to be translated in 026 format. Control characters are addressed to the card reader itself, and they remain in effect even when the reader is attached and subsequently detached.

The default condition can easily be changed from DEC026 to DEC029 by reading a 029 control card and then by saving the system with the MCR command SAV.

Table 9–5: Card Reader Control Characters

Punches	Columns	Meaning
12-11-0-1-6-7-8-9	1	EOF (alphanumeric)
12-11-0-1-6-7-8-9	(All 8 punches in the first 8 columns)	EOF (binary)
12-2-4-8	1	026-coded cards follow
12-0-2-4-6-8	1	029-coded cards follow

9.6 Card Reader Data Formats

The card reader reads data in either alphanumeric or binary format.

9.6.1 Alphanumeric Format (026 and 029)

Table 9–6 summarizes the translation from DEC026 or DEC029 card codes to ASCII.

9.6.2 Binary Format

In RSX–11M–PLUS binary format, the data are not packed, but are transferred exactly as read, one card column per word. Because each word has 16 bits and each card column represents only 12 bits, the data from the column are stored in the rightmost 12 bits of the word. The word's remaining 4 bits contain zeros.

9.7 Programming Hints

This section contains important information about programming the card reader driver described in this chapter. Section 9.4 contains information on operational error-recovery procedures that may be important for programming.

9.7.1 Input Card Limitation

Only one card can be read with a single QIO macro call. A request to read more than 80 bytes or columns, the length of a single card, does not result in a multiple card transfer. Only 80 columns are processed. It is possible to read fewer than 80 columns of card input with a QIO read function. For example, you can specify that only the first 10 columns of each card are to be read.

9.7.2 Aborting a Task

If a task waiting for the card reader to be readied is aborted, the card reader driver recognizes this fact within 1 second.

Table 9–6: Translation from DEC026 or DEC029 to ASCII

Character	Non-Parity ASCII	DEC029	DEC026	Character	Non-Parity ASCII	DEC029	DEC026
	173	12 0	12 0	'	054	0 8 3	0 8 3
	175	11 0	11 0	-	055	11	11
SPACE	040	None	None	.	056	12 8 3	12 8 3
!	041	12 8 7	12 8 7	/	057	0 1	0 1
"	042	8 7	0 8 5	0	060	0	0
#	043	8 3	0 8 6	1	061	1	1
$	044	11 8 3	11 8 3	2	062	2	2
%	045	0 8 4	0 8 7	3	063	3	3
AND	046	12	11 8 7	4	064	4	4
'	047	8 5	8 6	5	065	5	5
(	050	12 8 5	0 8 4	6	066	6	6
)	051	11 8 5	12 8 4	7	067	7	7
*	052	11 8 4	11 8 4	8	070	8	8
+	053	12 8 6	12	9	071	9	9
:	072	8 2	11 8 2	M	115	11 4	11 4
;	073	11 8 6	0 8 2	N	116	11 5	11 5
	074	12 8 4	12 8 6	O	117	11 6	11 6
=	075	8 6	8 3	P	120	11 7	11 7
>	076	0 8 6	11 8 6	Q	121	11 8	11 8
?	077	0 8 7	12 8 2	R	122	11 9	11 9
@	100	8 4	8 4	S	123	0 2	0 2
A	101	12 1	12 1	T	124	0 3	0 3
B	102	12 2	12 2	U	125	0 4	0 4
C	103	12 3	12 3	V	126	0 5	0 5
D	104	12 4	12 4	W	127	0 6	0 6

Table 9–6 (Cont.): Translation from DEC026 or DEC029 to ASCII

Character	Non-Parity ASCII	DEC029	DEC026	Character	Non-Parity ASCII	DEC029	DEC026
E	105	12 5	12 5	X	130	0 7	0 7
F	106	12 6	12 6	Y	131	0 8	0 8
G	107	12 7	12 7	Z	132	0 9	0 9
H	110	12 8	12 8	[	133	12 8 2	11 8 5
I	111	12 9	12 9	\	134	0 8 2	8 7
J	112	11 1	11 1	]	135	11 8 2	12 8 5
K	113	11 2	11 2	^	136	11 8 7	8 5
L	114	11 3	11 3	_	137	0 8 5	8 2

Chapter 10

QIO DEUNA Driver

10.1 Introduction to the QIO DEUNA Driver

For systems without the DECnet software, the RSX–11M–PLUS QIO DEUNA driver allows messages to be sent by using the DEUNA device. The DEUNA driver provides direct control over a line, allowing you to send data over a line to another system. To use the DEUNA driver, you issue the QIO$ macro to the XE device. The DEUNA driver is compatible with the DECnet software's Direct Line Access (DLX) interface, which permits easy migration to a DECnet system.

Use of the DEUNA driver requires a thorough knowledge of the MACRO–11 assembler and experience in writing real-time application programs. You must write tasks that synchronize with each other before transferring data. If tasks are not synchronized, the data can be lost during task-to-task communication. You must provide your own error-handling routines. The DEUNA driver software informs your task of any errors, but your task must be written to process error recovery. In addition, you must provide your own flow control over incoming messages to avoid message loss. Furthermore, applications must be designed so that adjacent nodes contain like routines for handling communications. For example, the driver does not, by itself, handle communications with DECnet nodes.

You can use QIO$s to communicate between your program and a program on an adjacent computer by using the Ethernet. In task-to-task communication between adjacent computers, the RSX–11M–PLUS QIO DEUNA driver is an efficient user of the central processing unit (CPU) and communication lines. You can build your own protocol that best suits the application.

Note

All messages are transmitted from your task's buffer. However, the driver buffers messages that it receives in a limited number of driver receive buffers. Therefore, you should make sure that at least two or more receive requests are outstanding at any given time to prevent messages from being lost. Unwanted messages are discarded.

A glossary of DEUNA terms is included at the end of this chapter.

10.1.1 Parameters That You Can Tailor

The parameters that you can tailor are as follows:

Parameter	Meaning
U$$NTS	Number of transmit ring entries (suggested three). On systems with UNIBUS Mapping Registers (UMRs) this parameter controls the number of UMRs the driver may use. For each transmission, the driver uses one UMR during the transfer.
U$$NRS	Number of receive ring entries (suggested eight).
U$$NPC	Number of ports per controller (suggested eight).
U$$NCT	Number of controllers.

10.1.2 Requirements for Tasks Using the RSX-11M-PLUS QIO DEUNA Driver

To run programs that use the DEUNA driver, the following conditions are required:

- The DEUNA driver must be loaded.
- The logical unit number (LUN) must be assigned to the XE device.

10.1.3 Special Considerations for Ethernet User Tasks

Externally, Ethernet devices appear to be single-line, point-to-point controllers (for example, UNA-0 and UNA-1). Internally, they are implemented as multipoint devices with each station representing an available port onto the Ethernet. Each driver supports eight ports. The limitation is due to the limited number of receive buffers available to the driver.

10.1.4 Messages on Ethernet

All messages on the Ethernet must include a destination address (48-bit) and a protocol type (16-bit). There are two modes that determine how messages are transmitted: physical address mode and multicast address mode.

Physical address mode defines a unique address for a single system on any Ethernet. Multicast address mode defines a multidestination address of one or more systems on a given Ethernet. With multicast addressing, any number of systems can be assigned a group address, so that they are all able to receive the same data in a single transmission.

Before transmitting and receiving messages, you must define a specific mode. You can do this by using the QIO$ IO.XSC macro, which sets characteristics. (See Section 10.3.2.)

10.1.5 Protocol and Address Pairs on Ethernet

Because the Ethernet allows multiuser tasks to access the physical link simultaneously, some way must be used to deliver received messages to the correct user task. To do this, each user task must enable unique protocol/address pairs to define which messages the task should receive. For example, user task 1 may enable protocol A to addresses 1 and 2, while user task 2 may enable protocol B to addresses 3 and 4. It is possible for two or more user tasks to enable the same protocol or addresses, providing that the protocol/address pairs are unique.

10.1.6 Opening Ethernet for Transmit and Receive

The Ethernet may be opened in the following three different modes (defined in EPMDF$):

Protocol	Mode	Meaning
LF$EXC	Exclusive	Your task has exclusive use of the specific protocol LF$EXC and no other user may transmit or receive using this protocol. (DECnet routing uses this mode.)
LF$DEF	Default	Your task should receive messages on the protocol LF$DEF, which would otherwise be discarded because there was no protocol/address pair set up.
Specified	Normal	You must specify the protocol/address pairs that are used for communications.

10.1.7 Padding Messages on Ethernet

You may select padding for an Ethernet message (LF$PAD) that prefixes the message with a 2-byte length field. The DEUNA pads the message out to the minimum Ethernet size on transmit. On receive, the length field indicates the amount of data present.

10.1.8 Hardware Errors on Ethernet

When a hardware error is detected on the Ethernet controller, all protocol/address pairings and multicast addresses are lost. After issuing the IO.XIN call to reinitialize the channel, you must reenable all protocol/address pairs and the multicast addresses.

10.2 DEUNA Driver QIO$s

Sections 10.2.1 to 10.2.4 describe some considerations for using QIO$ macros for the DEUNA driver.

10.2.1 Standards and Access to QIO$ Macros

The DEUNA driver conforms to normal RSX–11M–PLUS QIO$ standards. Standards for logical unit numbers (LUNs), event flags, I/O status blocks (IOSBs), asynchronous system traps (ASTs), and argument and parameter lists are observed. According to RSX–11M–PLUS standards, you may use any one of the three QIO$ formats. You may also use the QIO$ and Wait macro (QIOW$) to suspend further execution of the program until the call completes.

The macros are defined in the RSX–11M–PLUS macro library (EXEMC.MLB). This library is transferred to your system during system generation. The definitions and offsets that you use in the macros are contained in two definition macros, DLXDF$ and EPMDF$, in DEUNA.MLB.

You must issue .MCALL statements and explicitly invoke the macro in your MACRO-11 assembler program. An example follows:

```
.MCALL  DLXDF$,EPMDF$
   .
   .
   .
        DLXDF$
        EPMDF$
```

Table 10-1 summarizes the QIO DEUNA driver function codes and their meaning. Sections 10.3.1 to 10.3.7 describe each call, with its arguments and completion status codes.

10.2.2 Programming Sequence

Table 10-1 provides a list of the six steps required to transmit, receive, or read data on the Ethernet via the RSX-11M-PLUS QIO DEUNA driver.

Table 10-1: RSX-11M-PLUS QIO DEUNA Driver Function Codes and Their Meaning

Step	Code	Ethernet Operation
1	IO.XOP	Open the Ethernet device
2	IO.XSC	Set characteristics
3	IO.XTM	Transmit a message on the line
4	IO.XRC	Receive a message on the line
5	IO.XCL	Close the line
6	IO.XIN	Initialize the line after an unrecoverable hardware error.

Note

The IO.XTL control function loads DEUNA microcode. The driver support task, UML..., uses IO.XTL, which is a function you must not use.

10.2.3 Driver Installation

The system builds the driver at the time you perform a system generation.

To load the driver, enter the following MCR command:

```
>LOA XE:[/switches]
```

For RSX-11M-PLUS, you must perform the following additional steps to make the driver operational:

```
>CON SET XEA VEC=vvv CSR=xxxxxx
>CON ONLINE XEA
>CON ONLINE XE0:
>INS UML
```

Note

UML... is the microcode loader support task to the DEUNA driver (XEDRV). If you want the driver to bypass microcode loading, just remove the microcode support task (UML) from the system.

Make sure that the correct microcode file for the DEUNA driver is present on device LB in account [1,1].

10.2.4 QIO DEUNA Status Returns

Table 10–2 lists the status returns from QIO$ macros issued to the DEUNA driver.

Table 10–2: QIO DEUNA Driver Status Returns

	Value		
Code	**Decimal**	**Octal**	**Reason**
IS.SUC	1		The line has been opened successfully.
IE.ALN	-34	177736	The specified LUN is already in use.
IE.IFC	-2	177776	The specified LUN is not assigned to XE. For those characteristics blocks processed, return (XEDRV).
IE.NSF	-26	177646	Either you have entered an invalid controller identification format or the specified controller is not in the system.

10.3 QIO$ Macros

This section summarizes standard and device-specific QIO$ functions for the RSX–11M–PLUS QIO DEUNA driver.

10.3.1 IO.XOP—Open the Ethernet Device

You issue the QIO$ IO.XOP macro to open a line for direct line access, message transfer, and reception. The IO.XOP functions associate the specified LUN with the specified line. The line is then used when you issue further QIO$s for transmitting or receiving. The LUN must have been assigned to XE. To open the Ethernet device from the DEUNA driver, you issue this call using a device-ID string such as "UNA-0." The address of this string should be in p1. The driver scans its port database for an available port and assigns it to your task.

Format

```
QIO$   IO.XOP,lun,[efn],,[status],[ast], <p1,p2,p3>
```

Parameters

lun

Specifies the logical unit number associated with the line that you are opening.

efn

Specifies the optional event flag number set when the call completes.

status

Specifies the address of an optional 2-word status block that contains the completion status of the call in the low-order byte of the first word.

ast

Specifies the entry point into an optional AST routine, which you wrote, to be executed after this call completes.

p1

Specifies the address of an American Standard Code for Information Interchange (ASCII) string that identifies the controller on which the line is to be opened. The syntax of this string is as follows:

```
DEV-ctl
```

In this example, DEV (UNA) is the device mnemonic and ctl is the decimal value for the controller number.

p2

Specifies the length of the line identification field.

p3

Specifies the timeout value for the call. The timeout value is the amount of time that the receiver waits for a message to be transmitted. The low-order byte of the word designates the receive timeout value as follows:

Timeout = 0	For no receive timer.
Timeout = <n>	Where *n* is the timer value in seconds. The timer value *n* causes the timeout to have a range of n-1 to n. The high-order byte of this word is ignored.

10.3.2 IO.XSC—Set Characteristics (Ethernet)

You use this Ethernet QIO$ to set up the protocol/address pairs and multicast addresses. This function supplies a single characteristics buffer in parameters p1 and p2. This buffer may contain multiple characteristics blocks of the general format given in Section 10.3.2.1.

Format

QIO$ IO.XSC,lun,[efn],,[status],[ast], <p1,p2>

Parameters

lun

Specifies the logical unit number associated with the line that you are setting for a characteristics buffer.

efn

Specifies the optional event flag number set when the call completes.

status

Specifies the quantity processed on completion. The second word of the I/O status block (IOSB) indicates how much of the characteristics buffer has been processed.

ast

Allows the entry point in an optional asynchronous system trap (AST) routine, which you wrote, to be executed after this call completes.

p1

Specifies the address of the characteristics buffer.

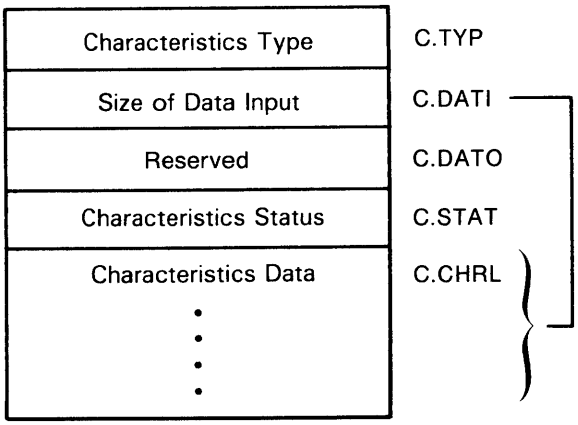
p2

Specifies the length of the characteristics buffer.

10.3.2.1 The Set Characteristics Buffer—General Format

The set characteristics buffer format may contain multiple characteristics blocks. Each characteristics block has the general format shown in Figure 10–1.

Figure 10–1: General Form of Characteristics Buffer



The fields in the general form of the characteristics block have the following meanings:

Field	Meaning
C.TYP	Indicates the type of characteristics being set.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set to indicate the success or failure of the characteristics function, for those characteristics blocks processed.
C.CHRL	Characteristic data.

Protocol flags are defined in EPMDF$ (LF$xxx).

Common error codes that are returned in C.STAT are as follows:

Error	Meaning
CE.UDF	Undefined function.
CE.RTS	Request too small (not enough data supplied).
CE.RTL	Request too large (too much data supplied).
CE.RES	Resource allocation failure.

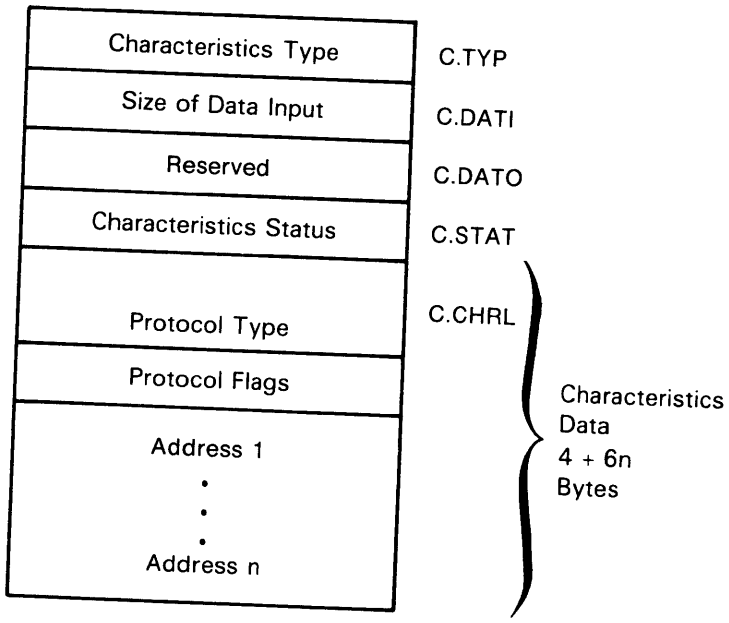
Note

The address field or fields should not be present if LF$EXC or LF$DEF is specified in the flags.

10.3.2.2 Set Characteristics—Setting Up Protocol/Address Pairs

Setting up protocol/address pairs allows transmission and reception of messages with the specified protocol to or from any of the addresses in the list. Figure 10–2 shows the characteristics buffer for this operation.

Figure 10–2: Buffer for Setting Up Protocol/Address Pairs



ZK-4087-85

The fields in the characteristics buffer for setting up protocol/address pairs have the following meaning:

Field	Meaning
C.TYP	Contains the characteristics type to set up protocol/address pairs: CC.DST = 200.
C.DATI	Indicates the size of data input—the number of bytes of characteristic data being supplied.
C.DATO	Unused for set characteristics.
C.STAT	Set, for those characteristics blocks processed, to indicate the success or failure of the characteristics function.
C.CHRL	Characteristics data.

Errors that are returned in C.STAT are shown in the following list:

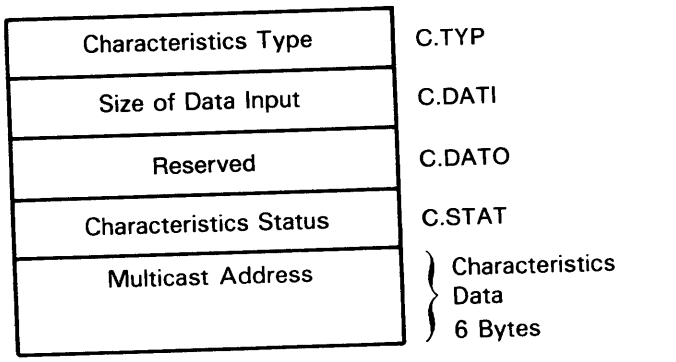
Error	Meaning
CE.PCN	Indicates protocol usage conflict due to one of the following: • Another user task has exclusive access to this protocol. • There is already a default task using this protocol, and this request is attempting to set up a new default user task. • The padding status of this protocol does not match what is requested.
CE.IUM	Indicates invalid use of multicast address; one of the addresses specified is multicast.
CE.ACN	Indicates address usage conflicts; the protocol/address pair is already in use.

10.3.2.3 Characteristics—Setting Up a Multicast Address

Setting up a multicast address allows reception of messages that are sent to the specified multicast address. The buffer for setting up a multicast address is shown in Figure 10–3.

The fields in the characteristics buffer for setting up a multicast address have the following meaning:

Figure 10–3: Buffer for Setting Up a Multicast Address



Field	Meaning
C.TYP	Contains the characteristics type to set up a multicast address: CC.MCT = 201.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set, for those characteristics blocks processed, to indicate the success or failure of the characteristics function.
C.CHRL	Characteristics data.

Errors returned in C.STAT are as follows:

Error	Meaning
CE.NMA	Not a multicast address.
CE.MCE	Multicast address already enabled.

10.3.3 IO.XTM—Transmit a Message on the Line

When your task transmits a message on the Ethernet, it must specify the destination address or the multicast address of this message along with the protocol type. It does specify the address if you put the parameters for the optional auxiliary characteristics buffer in parameters p3 and p4.

Format

QIO$ IO.XTM,lun,[efn],,[status],[ast], <p1,p2,p3,p4,[p5,p6]>

Parameters

lun

Specifies the logical unit number for the line on which you are transmitting data.

efn

Specifies the optional event flag number set when the call completes.

status

Specifies the address of an optional 2-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status in Section 10.3.3.3).

ast

Allows the entry point into an optional AST routine, which you wrote, to be executed after this call completes.

p1

Specifies the address of the buffer in your task that contains the message to be transmitted. Use the label specified in the DLXBUF macro call.

p2

Specifies the length of the message you are sending to the remote computer. Maximum buffer size is 1498 bytes.

p3

Specifies the address of the auxiliary characteristics buffer destination addresses.

p4

Specifies the length of the auxiliary characteristics buffer.

p5

Specifies the diagnostic buffer (see Section 10.4).

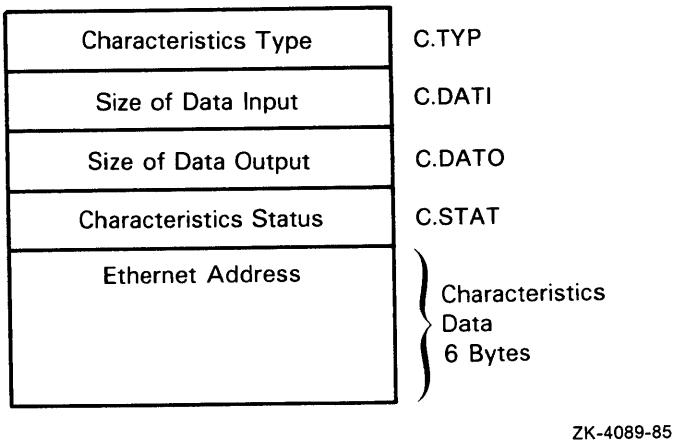
p6

Specifies the diagnostic buffer size (see Section 10.4).

10.3.3.1 Auxiliary Buffer to Set the Destination Address

To transmit on a line, you must first set up the auxiliary characteristics buffer with the Ethernet address. The auxiliary characteristics buffer has the same format as the set characteristics buffer described in Section 10.3.2.1. The buffer is shown in Figure 10–4.

Figure 10–4: Buffer for Setting the Ethernet Address



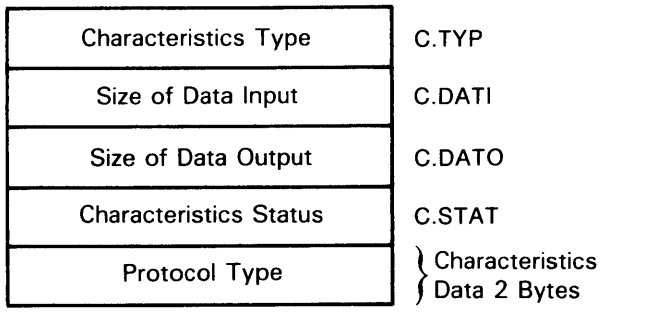
The fields in the auxiliary characteristics buffer for setting the Ethernet address have the following meaning:

Field	Meaning
C.TYP	Contains the characteristics type to set the Ethernet address: CC.ADR = 100.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set for those characteristics blocks processed, to indicate the success or failure of the characteristics function.
C.CHRL	Characteristics data.

10.3.3.2 Auxiliary Buffer to Set the Protocol Type

The protocol type must be transmitted along with the message. Use the auxiliary buffer shown in Figure 10–5 for this purpose.

Figure 10–5: Buffer for Setting the Protocol Type



The fields in the auxiliary characteristics buffer for setting the protocol type are as follows:

Field	Meaning
C.TYP	Contains the characteristics type to set the protocol type: CC.PRO = 101.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set, for those characteristics blocks processed, to indicate the success or failure of the characteristics function.
C.CHRL	Characteristics data.

Transmit requests on Ethernet channels must include an auxiliary characteristics buffer including both the destination address and protocol type. Failure to do so causes the transmit message to be returned with an IE.BAD error.

10.3.3.3 Completion Status Codes for IO.XTM

QIO$ IO.XTM returns the following completion status codes:

	Value		
Code	**Decimal**	**Octal**	**Reason**
IS.SUC	1		The message was transmitted to the remote system successfully.
IE.ABO	-15	177761	The transmission was aborted because an unrecoverable error occurred in the hardware device. When a message transmission completes with an IE.ABO code, the line is hung up. You must either issue a QIO$ IO.XIN to initialize the line (see Section 10.3.6) or close and reopen the line (see Sections 10.3.5 and 10.3.1, respectively) before you can use it again.
IE.IFC	-2	177776	The LUN is not assigned to XE.
IE.NLN	-37	177733	No line has been opened with the specified LUN.
IE.SPC	-6.	177772	The transmit buffer is too large or too small.

10.3.4 IO.XRC—Receive a Message on the Line

The QIO$ IO.XRC function receives a message on the Ethernet. When you receive a message on the Ethernet, you must find out the source address for this message along with the protocol type. You can do this by having an optional auxiliary characteristics buffer for receive messages in parameters p3 and p4.

Format

QIO$ IO.XRC,lun,[efn],,[status],[ast], <p1,p2,p3,p4,[p5,p6]>

Parameters

lun

Specifies the logical unit number associated with the line on which you receive the message.

efn

Specifies the optional event flag number set when the call completes.

status

Specifies the address of an optional 2-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status in Section 10.3.4.4).

ast

Allows the entry point in an optional AST routine, which you wrote, to be executed after this call completes.

p1

Specifies the address of the buffer in your task that receives the message.

p2

Specifies the length in bytes that you are allocating for the receive buffer. Maximum buffer size is 1498 bytes.

p3

Specifies the address of the auxiliary characteristics buffer.

p4

Specifies the length of the auxiliary characteristics buffer.

p5

Specifies the diagnostic buffer (see Section 10.4).

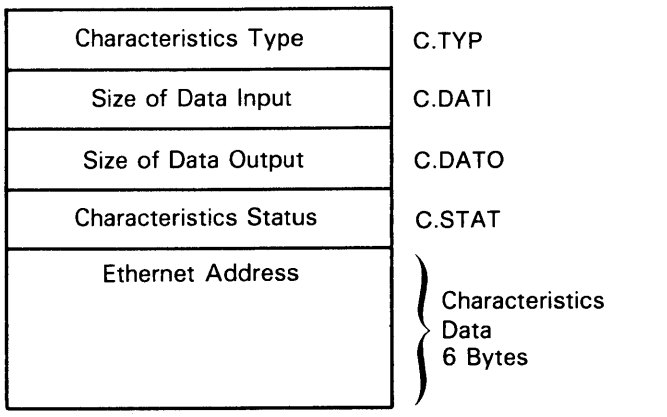
p6

Specifies the diagnostic buffer size (see Section 10.4).

10.3.4.1 Buffer for Reading the Ethernet Address

The auxiliary characteristics buffer has the same format as the set characteristics buffer described in Section 10.3.2.1. The buffer needed to provide for reading the Ethernet characteristics is shown in Figure 10–6.

Figure 10–6: Buffer for Reading the Ethernet Address



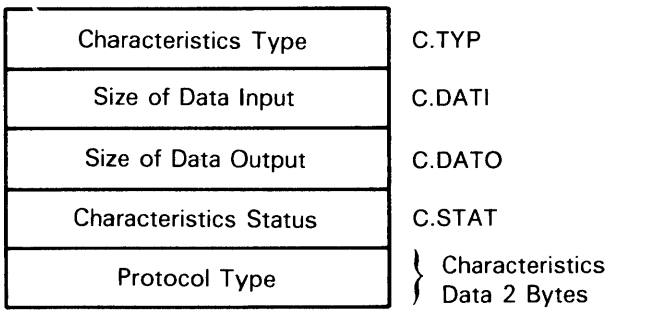
The fields in the auxiliary characteristics buffer for reading the Ethernet address are as follows:

Field	Meaning
C.TYP	Contains the characteristics type to read the Ethernet address: CC.ADR = 100.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set, for those characteristics blocks processed, to indicate the success or failure of the characteristics function.
C.CHRL	Characteristics data.

10.3.4.2 Buffer for Reading the Protocol Type

The buffer for reading the protocol type is shown in Figure 10–7.

Figure 10–7: Buffer for Reading the Protocol Type



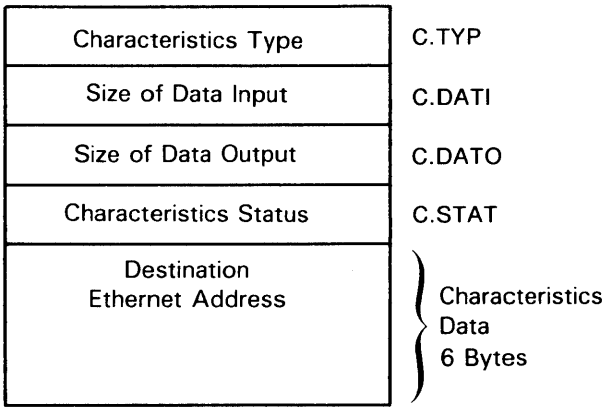
The fields in the auxiliary characteristics buffer for reading the protocol type are as follows:

Field	Meaning
C.TYP	Contains the characteristics type to read the protocol type: CC.PRO = 101.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set, for those characteristics blocks processed, to indicate the success or failure of the characteristics function.
C.CHRL	Characteristics data.

10.3.4.3 Buffer for Reading the Destination Ethernet Address

The buffer for reading the destination Ethernet address is shown in Figure 10–8:

Figure 10–8: Buffer for Reading the Destination Ethernet Address



The fields in the auxiliary characteristics buffer for reading the destination Ethernet address are as follows:

Field	Meaning
C.TYP	Contains the characteristics type to read the destination Ethernet address: CC.ADR = 102.
C.DATI	Indicates the size of data input (the number of bytes of characteristic data being supplied).
C.DATO	Unused for set characteristics.
C.STAT	Set, for those characteristics blocks processed, to indicate the success or failure of the characteristics function.
C.CHRL	Characteristics data.

10.3.4.4 Completion Status Codes for IO.XRC

QIO$ IO.XRC returns the following completion status codes:

Code	Value		Reason
	Decimal	Octal	
IS.SUC	1		You successfully received a message from the remote system. The second word of the I/O status block (IOSB) contains the number of bytes you actually received.
IE.ABO	-15	177761	The receive function was aborted because an unrecoverable error occurred in the hardware device. When a receive is aborted, the line is hung up. You must either issue QIO$ IO.XIN to initialize the line (see Section 10.3.6) or close and reopen the line (see Sections 10.3.5 and 10.3.1, respectively) before you can use it again.
IE.DAO	-13	177763	Either a message was received before a receive QIO$ was issued and the data is lost (this applies only to normal mode operations), or your task's buffer was too small to receive all the data. In the latter case, the message is truncated, and some data is lost. (The length of your task's buffer is contained in the second word of the IOSB.)
IE.IFC	-2	177776	The specified LUN is not assigned to XE.
IE.NLN	-37	177733	No line has been opened with the specified LUN.
IE.TMO	-74	177666	A timeout condition has occurred. No message was received within the timer interval specified when you opened or initialized the line.
IE.SPC	-6	177772	The transmit buffer is too large or too small.

10.3.5 IO.XCL—Close the Line

You issue the QIO$ IO.XCL macro to close an open line and stop the protocol.

Format

QIO$ IO.XCL,lun,[efn],,[status],[ast]

Parameters

lun

Specifies the logical unit number associated with the line that you are closing.

efn

Specifies the optional event flag number set when the call completes.

status

Specifies the address of an optional 2-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status in Section 10.3.5.1).

ast

Allows the entry point in an optional AST routine to be executed after this call completes.

10.3.5.1 Completion Status Codes for IO.XCL

QIO$ IO.XCL returns completion status codes as follows:

	Value		
Code	**Decimal**	**Octal**	**Reason**
IS.SUC	1		The line has been successfully closed.
IE.IFC	-2	177776	The specified lun is not assigned to XE.
IE.NLN	-37	177733	No line has been opened with the specified LUN.

10.3.6 IO.XIN—Initialize the Line

You issue the QIO$ IO.XIN macro to reinitialize a line after a fatal device error has occurred. When you use this QIO$, you must reset the mode and timer values.

Format

QIO$ IO.XIN,lun,[efn],,[status],[ast], <p1>

Parameters

lun

Specifies the logical unit number associated with the line that you are initializing.

efn

Specifies the optional event flag number set when the call completes.

status

Specifies the address of an optional 2-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status in Section 10.3.6.1.)

ast

Allows the entry point into an optional AST routine, which you wrote, to be executed after this call completes.

p1

Specifies the timer parameter. Use the same format as described for parameter p3 in IO.XOP. (See Section 10.3.1.)

10.3.6.1 Completion Status Codes for IO.XIN

IO.XIN returns completion status codes as follows:

Code	Value		Reason
	Decimal	Octal	
IS.SUC	1		The line has been successfully initialized.
IE.ABO	-15	177761	The initialization attempt has been aborted. A hardware device error or an attempt to initialize a line that did not require it could cause this problem.
IE.IFC	-2	177776	The specified LUN is not assigned to XE.
IE.NLN	-37	177733	No line has been opened with the specified LUN.

10.3.7 IO.XTL—Control Function

The QIO$ IO.XTL macro loads the microcode. IO.XTL is only valid when the driver is initializing the DEUNA controller. This function is a privileged function. Using it does not require a line to be open on the DEUNA.

Format

QIO$ IO.XTL+subfunction,lun,[efn],,[status],[ast]

Subfunctions

sub=0

Loads microcode to DEUNA memory.

sub=1

Ends load.

sub=2

Aborts load.

Parameters

lun

Specifies the logical unit number.

efn

Specifies the optional event flag number set when the call completes.

status

Specifies the address of an optional 2-word status block that contains the completion status of the call in the low-order byte of the first word (see completion status in Section 10.3.7.1). The second word is the count of the number of bytes loaded.

ast

Allows the entry point into an optional AST routine, which you wrote, to be executed after this call completes.

10.3.7.1 Completion Status Codes for IO.XTL

IO.XTL returns the following completion status codes:

	Value		
Code	**Decimal**	**Octal**	**Reason**
IS.SUC	1		The load function was successful.
IE.IFC	-2	177776	Invalid function.
IE.ABO	-15	177761	A hardware device error or an invalid buffer format could cause this error.
IE.SPC	-6	177772	The microcode ECO buffer is too large.
IE.PRI	-16	177760	Privilege violation.

10.4 Diagnostic Functions for IO.XTM/IO.XRC

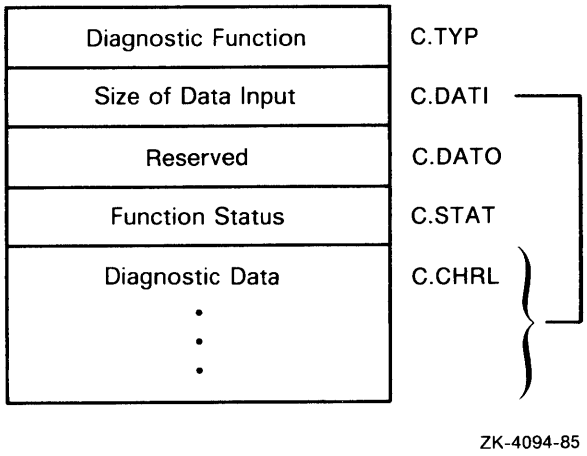
Your task may execute a number of the port control block functions of the DEUNA driver by using parameters p5 and p6. Parameter p5 is the address of the diagnostic buffer and parameter p6 is the size of the diagnostic buffer.

This buffer provides diagnostic hooks in the DEUNA driver. However, some of this buffer is needed for changing the DEUNA physical address, system ID, and so on.

Diagnostic function requests are passed to the driver in the same buffer format as the characteristics functions (see Section 10.3.2).

The diagnostic buffer format may contain multiple function request blocks. Each diagnostic request block is shown in Figure 10–9:

Figure 10–9: Diagnostic Request Block



Note

The status returned for the call does not reflect the status of the diagnostic functions in the diagnostic buffer. You must test the C.STAT word of each function request specified in the optional diagnostic buffer.

The valid function codes are noted in Table 10–3:

Table 10–3: Diagnostic Functions for IO.XTM/IO.XRC

Function Code	Meaning	Octal Buffer Size in Words
0	NOP function	0
2	Read default physical address	3
4	Read physical address	3
5	Write physical address	3
6	Read multicast list from UNA	36
12	Read counters	100
13[1]	Read and clear counters	100
14	Read UNA mode	1
15	Write UNA Mode	1
16	Read line status	10
17[1]	Read and clear line status	10
22	Read system ID	144 Maximum
23[1]	Write system ID	144 Maximum
24	Read load server address	3
25	Write load server address	3

[1]This function code must be issued by a privileged task.

Note

The buffer sizes specified are in addition to the 4-word header, that is, the function, input size, output size, function status, and data buffer.

10.5 Programming Hints

This section contains information on important programming considerations for tasks using the DEUNA driver described in this chapter.

10.5.1 Information on the DEUNA Device

In order to fully understand the hardware capabilities and programming features of the DEUNA, you should become familiar with the information contained in the *DEUNA User's Guide*.

10.5.2 DEUNA Read/Write Mode Function

To change the DEUNA mode, you should read the mode first and combine the change with the current mode by a logical OR function to prevent changing the other mode bits.

You cannot change the Transmit Message Pad Enable bit. The driver relies on the UNA to pad short messages. The driver reenables this bit each time it performs a Write Mode function.

10.5.3 DLX Incompatibility

The RSX–11M–PLUS DEUNA driver is not 100% compatible with the DECnet software's Direct Line Access (DLX). Under the DECnet software's DLX, the system ID, physical address, and mode are set by using Network Management. Therefore, you must set these three characteristics using the diagnostic functions (see Section 10.4).

10.5.4 Asynchronous I/O

The order of request completion is not preserved by the driver, because the driver has no way of knowing when a receive can be expected. Also, if you use diagnostic functions for a transmit or receive, requests without diagnostic functions may complete out of order. Therefore, you should use event flags to identify the request being completed.

10.5.5 Diagnostic Functions Without Data Transfer

To do diagnostic functions without data transfer, specify all the parameters correctly except for the size of the auxiliary characteristics buffer, which must be set to zero. This is an invalid buffer size and returns IE.SPC status for the call. However, the driver processes the diagnostic buffer, if it is present.

10.5.6 Maximum and Minimum Buffer Size

The maximum buffer size the DEUNA driver permits is 1500_{10} bytes. However, to provide the padding option described in Section 10.1.7, the maximum buffer size is 2 bytes less than the 1500_{10} bytes permitted by the DEUNA driver. The extra 2 bytes account for the byte count word in the transfer.

The minimum buffer size is 64 bytes. However, the driver does not check for a buffer size of less than 64; it assumes that the DEUNA driver always operates in padded mode. A small transmit buffer could result in transmitting 20 bytes and in receiving 64 bytes, because the DEUNA pads the buffer with zeros out to 64 bytes. Thus, the first 20 bytes will be data and the rest will be null bytes.

10.5.7 Default Mode

The driver initializes the DEUNA driver with the following mode bits set:

DEUNA Pads short transmit messages.

H4000 Collision test is enabled.

Note

The "Enable Half Duplex" mode bit should be set where it is not desirable for the DEUNA to receive its own transmissions.

10.5.8 Example of Connecting to a Remote Task

The following is a list of steps for a task to take to establish connection with a remote task by using the RSX-11M-PLUS QIO DEUNA driver:

1. Open a line on the Ethernet device.
2. Set characteristics as specified in Section 10.3.2. Setting characteristics establishes the protocol/address pairs that the system uses when it communicates with the remote systems on the network. For example, if your task communicates with multicast address 101,252,38 and DEUNA address 304,404,100 by using protocol 10000, the characteristics buffer would appear as follows:

```
.WORD  201        ; C.TYP   - Set multicast address (CC.MCT)
.WORD  6          ; C.DATI  - 6 bytes of address in buffer
.WORD  0          ; C.DATO  - Output data size 0 (none)
.WORD  0          ; C.STAT  - Characteristics status
.WORD  101        ; C.CHRL  - 1st word of multicast address buffer
.WORD  252        ;         - 2nd word of multicast address buffer

.WORD  38         ;         - 3rd word of multicast address buffer
.WORD  200        ; C.TYP   - Set protocol address pair
.WORD  10.        ; C.DATI  - 10. bytes of characteristics data
.WORD  0          ;         - Output data size 0 (none)
.WORD  0          ; C.STAT  - Characteristics status

.WORD  10000      ; C.CHRL  - Protocol type
.WORD  0          ;         - Protocol flags (normal)
.BYTE  304,0      ;         - 1st word of address
.BYTE  1,1        ;         - 2nd word of address
.BYTE  100,0      ;         - 3rd word of address
```

3. Once the protocol/address pairs are set, you should issue two or more receive QIO$s in anticipation of receiving a message on the Ethernet. In this way you can ensure that one request may be completing and still have another request outstanding to the driver. Upon completion of a receive request, your task must immediately issue another request before any other action to prevent received messages from being lost. Note that the driver discards unsolicited messages.
4. At this point, you may want to transmit a message to the participating systems, letting them know of your presence on the network. The format of such a message exchange is application dependent. Some sort of acknowledgment of the startup message may complete the startup sequence.

5. Now, you are ready to transmit and receive messages. If you receive an abort notification (IE.ABO) for a request, the line must be reinitialized via the QIO$ IO.XIN function before further activity can be resumed. Another way to reinitialize the line is to close it and reopen it. The auxiliary characteristics buffer for receives should have room for the address/protocol pair of the originating system as follows:

```
.WORD  100        ; C.TYP   - Read Ethernet address (CC.ADR)
.WORD  6          ; C.DATI  - 6 bytes of address in buffer
.WORD  0          ; C.DATO  - Output data size (6)
.WORD  0          ; C.STAT  - Characteristics status
.BYTE  0,0        ; C.CHRL  - Driver returns a
.BYTE  0,0        ;           3-word address
.BYTE  0,0        ;           in these three words

.WORD  101        ; C.TYP   - Read protocol type (CC.PRO)
.WORD  2          ; C.DATI  - 2 bytes of protocol type
.WORD  0          ; C.DATO  - Output data size (2)
.WORD  0          ; C.STAT  - Characteristics status
.WORD  0          ; C.CHRL  - Contains protocol type
```

6. The auxiliary characteristics buffer for transmits must contain the destination address/protocol pair as follows:

```
.WORD  100        ; C.TYP   - Set Ethernet address (CC.ADR)
.WORD  6          ; C.DATI  - 6 bytes of address in buffer
.WORD  0          ; C.DATO  - Output data size 0
.WORD  0          ; C.STAT  - Characteristics status
.BYTE  304,0      ; C.CHRL  - Task must pass
.BYTE  1,1        ;           3-word address
.BYTE  100,0      ;           in these three words

.WORD  101        ; C.TYP   - Set protocol type (CC.PRO)
.WORD  2          ; C.DATI  - 2 bytes of protocol type
.WORD  0          ; C.DATO  - Output data size 0
.WORD  0          ; C.STAT  - Characteristics status
.WORD  10000      ; C.CHRL  - Must contain protocol type
```

7. Upon completion of Ethernet I/O, issue a close (IO.XCL) to release the line.

10.6 Glossary

Controller

A single piece of peripheral equipment of the system bus that communicates with one or more external devices. A single DEUNA is a controller.

CSR

The control and status registers for a controller. These are the ports through which the driver communicates with the device.

DEUNA

DIGITAL Equipment UNIBUS Network Adapter.

DLX

Direct Line Access (DLX) controller. Enables programs to have a direct, high-level interface to a physical line on systems with DECnet support.

DNA

DIGITAL Network Architecture. A network architecture of protocols, interfaces, and functions that enable DECnet network nodes to communicate.

Line

A communication path to another system. For example, a port on the Network Interconnect (NI) is a line.

Multiaccess channel

The Ethernet is unlike other data links supported by DIGITAL communications software products in that more than one user task may use a single circuit simultaneously.

NI

Network Interconnect is the group of DECnet products that implement the Xerox,[1] Intel,[2] and DEC intercompany Ethernet specifications.[3]

Protocol type

A unique 16-bit address that distinguishes each user task of the NI.

---

1 Xerox is a registered trademark of the Xerox Corporation

2 Intel is a trademark of the Intel Corporation

3 This function code must be issued by a privileged task.

Chapter 11

PCL11 Parallel Communications Link Drivers

11.1 Introduction to the PCL11 Parallel Communications Link Driver

PCL11 Parallel Communications Link hardware is supported on RSX–11M–PLUS systems by two drivers. One driver supports the transmitter function and the other driver supports the receiver function. The PCL11-B is a hardware interface that functions as a time division multiplexed (TDM) interface over which several PDP–11 computers can transfer data to each other. Each PCL11-B consists of a transmitter, receiver, and master section. The transmitter section can transfer parallel 16-bit words along the TDM bus to a receiver section of a separate PCL11-B on a different PDP–11 computer's UNIBUS. One of the PCL11-B units attached to the TDM bus must have its master section enabled to effect the data transfer.

11.1.1 PCL11-B Hardware

Each PCL11-B transmitter and receiver section has a unique TDM bus address (hardware configured). When a master section is enabled, it places a transmitter address on the TDM bus for a period of time, called a timeslice. During the timeslice, the addressed transmitter can address the desired receiver section and transmit one word; the transmitter waits for the receiver to acknowledge the word or an indication that the word was not accepted. If the word is not accepted, it normally retransmits the word on the next available timeslice. Thus, a message up to 32K words long can be transmitted to a receiver one word at a time during the time in which other similar TDM transactions are multiplexed for other PCL11-B devices.

11.1.2 PCL11 Transmitter Driver

The PCL11 transmitter driver provides two basic functions. First, it must receive data sent by the attached task and store it in a buffer in the PCL11 hardware. Then, the driver passes proper receiver address and command information to the PCL11 transmitter hardware to effect the actual transfer over the TDM bus.

11.1.3 PCL11 Receiver Driver

The PCL11 receiver driver also performs two basic functions. First, it must remove data from the receiver buffer and send it to the connected task. In addition, the receiver driver must acknowledge a transmitter when a data transmission is requested by that transmitter. Subsequent requests by other transmitters on the TDM bus are ignored until all message transactions with the current transmitter are completed.

11.2 Get LUN Information Macro

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the following information for the PCL11 transmitter and receiver drivers. A setting of 1 indicates that the described characteristics are true for PCL11 transmitter and receiver drivers.

Bit	Setting	Meaning
0	1	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	File-structured device
4	0	Single-directory device
5	1	Sequential device
6	0	Mass-storage device
7	0	User-mode diagnostics supported
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo device
13	0	Device mountable as a communications channel
14	0	Device mountable as a Files–11 volume
15	0	Device mountable

Word 3 contains device driver-specific information, as follows:

Transmitter driver

The low byte of word 3 contains the number of transmit retries remaining after completing the current data transmit function if the current data transmit function attempt is not accepted by the addressed receiver. The high byte of word 3 is undefined.

Receiver driver

The low byte of word 3 contains the index of the current state of the receiver driver. Use these states primarily for diagnostic purposes as they are defined as follows:

Index Value	Meaning
0	No task is connected.
+2	Task is connected but not triggered.
+4	Task is triggered and waiting for IO.RTF or IO.ATF function.
+6	Task triggered and timed out while waiting for IO.RTF or IO.ATF function.
-2	IO.ATF function is in progress.
-4	Task is connected, not triggered, and has an IO.ATF function in progress.
-6	An IO.RTF function is in progress.

The high byte of word 3 is undefined. Word 4 is undefined. Word 5 is the default buffer size in bytes. For the PCL11, this value is 64 bytes.

11.3 QIO Macro—PCL11 Transmitter Driver Functions

The following sections describe both the standard and device-specific QIO functions.

11.3.1 Standard QIO Functions

Table 11–1 lists the standard functions of the QIO macro that are valid for the PCL11 transmitter driver.

Table 11–1: Standard QIO Functions for PCL11 Transmitters

Format	Function
QIO$C IO.ATT,...	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O request

11.3.2 Device-Specific QIO Functions

Table 11–2 lists the device-specific functions of the QIO macro that are valid for the PCL11 transmitter driver.

Table 11–2: Device-Specific QIO Functions for PCL11 Transmitters

Format	Function
QIO$C IO.ATX,..., <stadd,size, flagwd,id,retries,retadd>	Attempt message transmission
QIO$C IO.SEC,...,	Sense master section status
QIO$C IO.STC,..., <stadd,size, [state],[mode],,retadd>	Set master section characteristics

Parameters

stadd

Specifies the starting address of a data buffer. (Its description and function are dependent upon the specific QIO function.)

size

Specifies the data buffer size in bytes. (Its description and function are dependent upon the specific QIO function.)

flagwd

Specifies the value of the flagword that is to precede the message being sent. The flags specify the desired receiver function as defined by your task's protocol.

id

Specifies the identifier of the central processing unit (CPU) to which the message is to be sent. This identifier is the desired receiver's TDM bus address. It appears in the high byte of the first word of the master section I/O status block (IOSB). The identifier number is an octal value contained in the high byte of the parameter word. For example, receiver number 1 is specified as 400, receiver number 2 is specified as 1000, and so forth.

retries

Specifies the number of retries that are attempted, following the first attempt, before returning error status to the calling task. Retries occur because of the following conditions:

- The first attempt was unsuccessful.
- Transmission errors occurred.
- A master down condition occurred.

retadd

Specifies the starting address of a 7-word buffer into which the contents of the six transmitter registers and the Transmitter Master/Maintenance Register (TMMR) are moved prior to returning to the calling task. Information describing the contents of these registers can be obtained by referring to the hardware documentation supplied with the PCL11 option.

state

Specifies the desired state setting for the transmitter, as follows:

Parameter	Specified State
SS.MAS	TDM bus master
SS.NEU	Neutral (default state)

mode

Specifies the desired mode setting for allocating transmitter timeslices on the TDM bus, as follows:

Parameter Entered	Mode
MS.AUT	Autoaddressing (default mode)
MS.ADS	Address silo

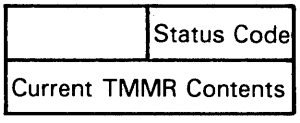
11.3.2.1 IO.ATX

The IO.ATX I/O function requests an attempt to transmit a message to a specified CPU. The message to be transmitted is contained in a data buffer starting at the address specified in the stadd parameter. This address must be on a word boundary. The data buffer size specified in the size parameter must be an even, positive value. The flagword parameter contains information, which you defined, that the receiving task uses to determine whether to accept or reject the message. The id parameter is the receiver TDM bus address. The task uses this address to direct a message to a specific CPU. Other parameters are as described in Section 11.3.2.

11.3.2.2 IO.SEC

The IO.SEC function senses the master section status. Upon successful completion of this function, the IOSB contains a typical I/O status code (IS.SUC) return in the low byte of the first word, and current TMMR contents in the second word, as shown in Figure 11–1.

Figure 11–1: IO.SEC Status Block Contents



ZK-5881-HC

Note

The optional isb parameter (see Chapter 1) must be included in this QIO request.

11.3.2.3 IO.STC

The IO.STC function sets the master section operational characteristics. IO.STC can be issued only by a privileged task. Correct use of the function depends upon the current (or specified) operating state of the master section and proper use of parameters. Use each parameter as described in the following paragraphs. Refer to all parameters in the sequence shown for a correct interpretation of parameter usage.

Parameters

state

Determines the overall function of the master section (and transmitter and receiver sections) in the PCL11 communications link as it relates to the TDM bus. The neutral state (SS.NEU) places the master section in an inactive state where the unit sends and receives messages in a normal manner, but the master section cannot control transmitter timeslice allocation on the TDM bus. The master state (SS.MAS) designates this unit as TDM bus master, enabling control of transmitter unit timeslice allotments on the TDM bus; only one master section on the TDM bus can be designated TDM bus master.

mode

Allows the TDM bus master to allocate transmitter timeslices in one of two ways: autoaddress mode (MS.AUT) or address silo mode (MS.ADS). When operating in the autoaddress mode (MS.AUT), which is the default mode for the TDM bus master, equal timeslice allotments are given to each transmitter unit; transmitter unit addresses are sequentially put on the TDM bus in descending order, one address for each timeslice. When operating in the address silo mode, transmitter unit addresses are transmitted in a sequence, which you specified, allowing up to 50% of the timeslices to be allocated to one transmitter unit, if desired.

The actual sequence of transmitter timeslice allocations for the address silo mode is set up in your task data buffer referenced by the stadd parameter. Certain constraints must be observed when specifying this information, as follows:

- Each entry in the buffer must be a byte containing a transmitter unit address.
- At least 20 entries, but not more than 50 entries, must be specified. If less than 20 entries are specified, the driver repeats the entire sequence, as specified, to attain the required minimum of 20 addresses. If more than 50 addresses are specified, no change in timeslice allocation is effected and an IE.VER error status is returned to the task.
- Identical transmitter addresses in either adjacent bytes or in first and last bytes should be avoided. When identical addresses appear in adjacent bytes in this manner, the driver inserts invalid "pad" transmitter addresses between identical addresses, effectively resulting in no-operation timeslices.
- Transmitter addresses must be decimal values ranging from 1 to 32 (inclusive) that correspond to addresses implemented on the actual transmitter unit hardware.
- The size parameter must correctly specify the number of address bytes contained in the buffer referenced by the stadd parameter.

11.4 PCL11 Transmitter Driver Status Returns

Table 11–3 lists PCL11 transmitter driver return status codes and probable reasons.

Table 11–3: PCL11 Transmitter Driver Status Returns

Code	Reason
IS.SUC	**Successful completion** The QIO function was successfully completed. If an IO.ATX function was completed, the second status word contains the number of bytes transferred; the message was not truncated. If an IO.SEC function was completed, the second status word contains the current contents of the master section's TMMR.
IS.TNC	**Successful transfer but message truncated** The IO.ATX function was completed, but the message was truncated by the receiver (the receiver buffer is too small). The transmitter unit cannot determine how many words were actually received by the receiver unit; the second word of the I/O status block (IOSB) contains the length of the requested transfer, rather than the actual count of words successfully received in the receiver's buffer.
IE.BAD	**Bad parameter specification** A bad parameter specification was included in the IO.ATX function, or an invalid state parameter or TDM bus timeslice allocation addressing mode was specified in the IO.STC function. This error status is also returned when an IO.STC function, issued to a TDM bus master operating in the address silo mode, refers to a data buffer containing an illegal series of transmitter addresses. An illegal series of addresses occurs when the number of entries specified for the timeslice allocation, plus the required number of pad addresses, either exceeds 50 or is less than 0.
IE.DNR	**Device not ready** This error status return occurs in response to an IO.ATX function when one of the following occurs: • Power failure occurs in this central processing unit (CPU). • Device timeout occurs (no response from the addressed receiver). • Receiver is too slow in accepting or rejecting the transfer request. • The master section is inoperative. This error status is returned only after the number of retries specified in the IO.ATX function have been attempted without success.
IE.VER	**Unrecoverable error** The IO.STC function state setting could not be achieved because the task is not privileged or another device is TDM bus master.
IE.SPC	**Illegal user task buffer** The buffer address specified in the IO.ATF function is outside of the issuing task's address space.

Table 11–3 (Cont.): PCL11 Transmitter Driver Status Returns

Code	Reason
IE.REJ	**Transfer rejected** The data transfer request specified in the IO.ATX function was rejected by the addressed receiver based on the source CPU identifier of the task issuing the request and flagword.
IE.FLG	**Event flag already specified** An event flag was previously specified in an IO.STC function.
IE.BBE	**Transmission error** This error status is returned only after the number of retries specified in the IO.ATX function have been attempted without a successful transmission. (Cyclic redundancy check (CRC) errors or parity errors have been detected on each attempt.)
IE.ABO	**Request terminated** This status is returned when a pending I/O function has been aborted in response to an IO.KIL function being issued by the task.
IE.IFC	**Illegal function code** A function code was specified in an I/O request that is illegal for PCL11 transmitters.

11.5 QIO Macro—PCL11 Receiver Driver Functions

The following sections list the standard and device-specific QIO functions for driver to the PCL11 receiver.

11.5.1 Standard QIO Functions

Table 11–4 lists the standard function of the QIO macro that is valid for the PCL11 receiver driver.

Table 11–4: Standard QIO Functions for PCL11 Receivers

Format	Function
QIO$C IO.KIL,...	Cancel I/O request

11.5.2 Device-Specific QIO Functions

Table 11–5 lists the device-specific functions of the QIO macro that are valid for the PCL11 receiver driver.

Table 11–5: Device-Specific QIO Functions for PCL11 Receivers

Format	Function
QIO$C IO.CRX,..., <tef,bufadd>	Connect for reception
QIO$C IO.RTF,...	Reject transfer
QIO$C IO.ATF,..., <stadd,size,retadd>	Accept transfer
QIO$C IO.DRX,...	Disconnect for reception

Parameters

tef

Specifies the number of a "trigger" event flag that is set whenever a flagword is received over the TDM bus.

bufadd

Specifies the address of a 2-word buffer containing the transmitter ID, trigger status, and the flagword.

stadd

Specifies the address of a data buffer to receive the message. This address must occur on a word boundary (even address).

size

Specifies the data buffer size in bytes. The size specified must be an even, positive value.

retadd

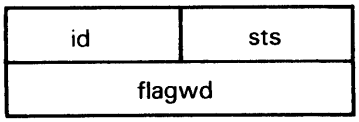
Specifies the address of a 6-word buffer into which the contents of the six PCL11 receiver hardware registers are returned upon successful completion of the function. Information describing the contents of these registers can be obtained by referring to the hardware documentation supplied with the PCL11 option.

11.5.2.1 IO.CRX

The IO.CRX function connects the issuing task to the receiver if the receiver is not currently connected to another task. When connected, this task is the only task capable of receiving messages by means of the receiver on this CPU. The trigger event flag (a local, common, or group-global event flag) informs the task when a message is pending. It is set when a flagword is received over the TDM bus. When this happens, a significant event is declared and the connected task is considered "triggered." The flagword is the first word transmitted by a transmitter when it attempts to send a message to the receiver unit.

The bufadd parameter must be included in this I/O function to specify the address of a 2-word block, as shown in Figure 11–2.

Figure 11–2: IO.CRX Status Block Contents



ZK-5882-HC

In Figure 11–2, sts, id, and flagwd are defined as follows:

sts
Specifies the current trigger status.

id
Specifies the identification code of the transmitter attempting to send the message.

flagwd
Specifies the flagword transmitted to the connected receiver.

Based on the information contained in the flagword and the identification code of the transmitter unit, the task can accept or reject the transfer. (Two I/O functions are provided for this purpose; see Sections 11.5.2.2 and 11.5.2.3) The receiver must respond to the transmitter's request within approximately 1.5 seconds; otherwise, an IE.DNR error status is returned to the task attempting the transmission.

11.5.2.2 IO.RTF

The IO.RTF function informs the transmitter device that the message is being rejected by the receiver. Any attempt to issue this I/O function when the trigger event flag is not set is ignored, and an IE.NTR error status is returned to the task.

11.5.2.3 IO.ATF

The IO.ATF function informs the transmitter device that the message is being accepted. Parameters specify both the data buffer into which the received data is transferred and the 6-word buffer that receives the contents of the receiver section hardware registers upon successful completion of the function.

Unlike the IO.RTF function, the IO.ATF function can be issued before the task is triggered. When this process is used, the IO.ATF function is queued for reception of any flagword. When the flagword is received, the receiver driver immediately executes the IO.ATF function; the connected task is not triggered and the flagword is not made available to the task. This approach is useful when it is not necessary to examine flagwords or to accept messages based on the source.

11.5.2.4 IO.DRX

This function is issued by a task to disconnect the receiver for use by other tasks.

11.6 PCL11 Receiver Driver Status Returns

Table 11–6 lists PCL11 receiver driver return status codes and probable reasons.

Table 11–6: PCL11 Receiver Driver Status Returns

Code	Reason
IS.SUC	**Successful completion** The I/O function or triggering of the task was completed successfully. When this status is returned upon completion of the IO.ATF function, the high-order byte of the first word in the I/O status block contains (IOSB) the identification code of the transmitter device that sent the flagword. The second word of the IOSB contains the number of bytes transferred over the TDM bus. When this status is returned as a result of an IO.CRX function, and the task being triggered, the IOSB contains information that enables the task to accept or reject the message (see Section 11.5.2.1).
IS.TNC	**Successful transfer but message truncated** This I/O status code is returned when the message is terminated because the receiver task message buffer specified in the IO.ATF function is too small to contain the message being received. The second word of the I/O status word contains the number of bytes successfully transferred.
IE.BAD	**Bad parameter specification** A bad parameter specification was included in the requested function.
IE.DNR	**Device not ready** This error status return occurs in response to an IO.RTF or IO.ATF function when one of the following occurs: • Power failure occurs in this central processing unit (CPU). • Device timeout occurs (no response from addressed receiver). • Receiver is too slow in accepting or rejecting the transfer request. • The master section is inoperative.
IE.SPC	**Illegal user task buffer** The buffer address specified in the IO.ATF function is outside of the issuing task's address space.
IE.DNA	**Task not connected for reception** The requested function cannot be executed because the task is not connected to the receiver.
IE.DAO	**Data overrun** This I/O status code is returned when the task is triggered, but the previous transfer request has neither been accepted nor rejected. When the task issues an IO.RTF or IO.ATF function, that function applies to the new (most recent) flagword; the previous request is ignored.

Table 11–6 (Cont.): PCL11 Receiver Driver Status Returns

Code	Reason
IE.DAA	**Device already connected for reception** This I/O status code is returned in response to the IO.CRX function when the receiver is already connected to this task or any other task. No operation is performed.
IE.NTR	**Task not triggered** This I/O status code is returned when a task attempts to issue an IO.RTF function prior to the task being triggered.
IE.BBE	**Transmission error** This error status is returned when an IO.ATF function is in progress and a cyclic redundancy check (CRC) error or parity error has been detected.
IE.ABO	**Request terminated** This status code is returned when a pending I/O function has been aborted in response to an IO.KIL function being issued by the task.
IE.FHE	**Fatal hardware error** The requested function cannot be executed because of a hardware failure.
IE.IFC	**Illegal function code** A function code was specified in an I/O request that is illegal for PCL11 transmitters.

Chapter 12

Laboratory Peripheral Accelerator Driver

12.1 Introduction to the Laboratory Peripheral Accelerator Driver

The Laboratory Peripheral Accelerator driver (LPA11-K) is an intelligent, direct memory access (DMA) controller for DIGITAL's laboratory data acquisition I/O devices. It is a fast, flexible, and easy-to-use microprocessor subsystem that allows analog data acquisition rates up to 150,000 samples per second. The LPA11-K is for applications requiring concurrent data acquisition and data reduction at high rates.

The LPA11-K is supported through a device driver and a set of program-callable routines. The device driver supports multiple controllers and can be configured as resident or loadable. The program-callable support routines are linked with your task at task-build time. These routines are highly modular. Therefore, a particular task need only contain that code necessary for the facilities that it actually uses.

The LPA11-K operates in two distinct modes: dedicated and multirequest. The subsections that follow summarize each mode.

12.1.1 LPA11-K Dedicated Mode of Operation

In dedicated mode, only one task (that is, one request) can be active at a time and only analog I/O data transfers are supported. Up to two analog converters can be controlled simultaneously. Sampling is initiated by an overflow of the real-time clock or by an externally supplied signal.

12.1.2 LPA11-K Multirequest Mode of Operation

In multirequest mode, sampling from all device types is supported. Up to eight user tasks can be simultaneously active. The sampling rate for each user task is a multiple of the common real-time clock rate. Independent rates can be maintained for each task. Both the sampling rate and the device type are specified as part of each data transfer request.

12.2 Get LUN Information Macro

If a Get LUN Information system directive is issued for a logical unit number (LUN) associated with an LPA11-K, word 2 (the first characteristics word) contains all zeros, words 3 and 4 are undefined, and word 5 contains a 16-bit buffer preset value that controls the rate of the real-time clock interrupts.

12.3 The Program Interface

A collection of program-callable subroutines provides access to the LPA11-K. The formats of these calls are fully documented here for FORTRAN programs. MACRO–11 programmers access these same subroutines either through the standard subroutine linkage or through the use of two special-purpose macros. Optionally, MACRO–11 users can control an LPA11-K directly by using device-specific QIO functions. Both FORTRAN and MACRO programs must contain at least one I/O status block (IOSB) for retrieval of status information. The following subsections, therefore, describe:

- The FORTRAN interface
- The MACRO–11 interface
- The IOSB

Note

The subroutines documented in this chapter represent the high-level interface to the LPA11-K. Using these subroutines requires an understanding of hardware capabilities, configuration details, and hardware status codes as described in the *LPA11-K Laboratory Peripheral Accelerator User's Guide*.

12.3.1 FORTRAN Interface

Table 12–1 lists the FORTRAN interface subroutines for accessing the LPA11-K.

The calling sequences of the routines listed in Table 12–1 are compatible with the K-series support routines, described in Chapter 13, except as noted. The following subsections briefly describe the function and format of each FORTRAN subroutine call.

Table 12–1: FORTRAN Subroutines for the LPA11-K

Subroutine	Function
ADSWP	Initiate synchronous analog-to digital (A/D) sweep
CLOCKA	Set Clock A rate
CLOCKB	Control Clock B
CVADF	Convert A/D input to floating point
DASWP	Initiate synchronous digital-to-analog (D/A) sweep
DISWP	Initiate synchronous digital input sweep
DOSWP	Initiate synchronous digital output sweep

Table 12–1 (Cont.): FORTRAN Subroutines for the LPA11-K

Subroutine	Function
FLT16	Convert unsigned integer to a real constant
IBFSTS	Get buffer status
IGTBUF	Return buffer number
INXTBF	Set next buffer
IWTBUF	Wait for buffer
LAMSKS	Set masks buffer
RLSBUF	Release data buffer
RMVBUF	Remove buffer from device queue
SETADC	Set channel information
SETIBF	Set array for buffered sweep
STPSWP	Stop sweep
XRATE	Compute clock rate and preset

12.3.1.1 ADSWP—Initiate Synchronous A/D Sweep

The ADSWP routine initiates a synchronous analog-to-digital (A/D) input sweep through an LPS-11 or an AD11-K (and, if present, the AM11-K).

If differential input is desired for the AD11-K/AM11-K, the channel increment must be set to 2 by calling the SETADC routine. The default channel increment is 1 (single-ended input).

Format

CALL ADSWP (ibuf,lbuf,[nbuf],[mode],[idwell],[iefn],[ldelay],[ichn],[nchn],[ind])

Parameters

ibuf

Specifies a 40-word array initialized by the SETIBF routine. The first two words of the array are the IOSB.

lbuf

Specifies the size, in words, of each data buffer. All data buffers must be equal in size and lbuf must be greater than 5. In dedicated mode, lbuf must be at least 257 words.

nbu

Specifies the number of buffers to be filled. If nbuf is omitted or set equal to 0, indefinite sampling occurs. The STPSWP routine terminates indefinite sampling.

mode

Specifies the sampling options. The default is 0. The mode bit values listed below that are preceded by a plus sign (+) are independent and can be added or ORed together (assuming that the sampling options are applicable to the mode of operation). Those values not preceded by a plus sign are mutually exclusive and the task can use only one such value at a time. All bit values not listed below are reserved.

The following values can be specified:

0 Absolute channel addressing (default). This mode allows your task to directly access all 64 channels of an A/D converter.

+32 Dual A/D conversion serial/parallel. This option applies to dedicated mode only. It is ignored in multirequest mode.

+64 Multirequest mode. If this value is not specified, the request is for dedicated mode. If the request mode does not match the mode of the hardware (that is, different microcode in the master microprocessor), the LPA11-K rejects the request with an appropriate error code.

+512 External trigger (ST1). Use this mode when you want to use your own external sweep trigger. The external trigger is supplied by a jumper connecting the AD11-K External Start input to the KW11-K Schmitt Trigger 1 output. You can use this external trigger connection only in dedicated mode. If you select mode 512, your task must specify a Clock A rate of -1 for proper A/D sampling. This is non-clock-driven sampling.

+1024 Time stamp with Clock B (multirequest mode only).

+2048 Event marking (multirequest mode only). LAMSKS must be called to specify an event mark channel and event mark mask.

+4096 Start method. If set, digital input start. If clear, immediate start. LAMSKS must be called to specify a digital start channel and digital start mask (multirequest mode only).

+8192 Dual A/D converter (dedicated mode only).

+16384 Data overrun NONFATAL/FATAL. If selected, data overrun is considered nonfatal. The LPA11-K defaults to fill buffer 0. (See Section 12.4 for a discussion of buffer management.)

idwell

Specifies the number of clock overflows (pulses) between data sample sequences. As an example, if idwell is 20 and nchn is 3, the following occurs: after 20 pulses, 1 channel is sampled on each of the next 3 pulses. Then, no sampling takes place for the next 20 pulses. In multirequest mode, this facility permits different sample rates for the same hardware clock rate and preset. In dedicated mode, the clock hardware rate controls sampling and this idwell parameter is ignored.

If compatibility with K-series support routines is desired, your task must first establish the clock preset by calling the CLOCKA routine. The sweep start command uses the default idwell value of 1. For the K-series, this procedure sets the rate as desired.

Note

This parameter is called iprset in the K-series support routines described in Chapter 13. Its function is different from the idwell parameter described here.

iefn

Specifies the event flag number (1 to 28, 30 to 96), the name of a completion routine, or 0. If you use 0 or default this value, the driver uses event flag 30 for internal synchronization. If you select an event flag with iefn, the driver sets the selected event flag as each buffer is filled. Note that the LPA11-K support routines reserve event flag 29 for internal synchronization. If iefn is greater than 96, the driver considers it to be a completion routine that is called with a JSR PC. Such routines must return with an RTS PC (or a FORTRAN RETURN statement).

FORTRAN completion routines must not contain any of the following:

- Any I/O through the FORTRAN run-time system
- Any use of virtual arrays
- Any use of floating-point operations
- Any errors, because error reporting is done through the FORTRAN run-time system
- Anything else that may change the FORTRAN impure area

Any of the above may result in fatal task errors or unpredictable results.

If multiple sweeps are initiated, your task should specify different event flags. Adherence to this limitation cannot be enforced by the software.

idelay

Specifies the delay from the start event (DR11-K) until the first sample in irate units. This feature is supported in multirequest mode only. Default or 0 indicates no delay.

ichn

Specifies the number of the first channel to be sampled. The default of 0 applies only if ichn was not established in a prior call to the SETADC routine.

nchn

Specifies the number of channels to sample. The default is 1. The nchn parameter may be set up with the SETADC routine. The number of channels specified are sampled at a rate of 1 per clock interrupt. If nchn equals 1, the single channel bit is set in the mode word of the start Request Descriptor Array (RDA).

ind

Receives a success or failure code as follows:

1 Indicates that the sweep was initialized successfully.

0 Indicates an illegal argument list, or SETIBF was not called prior to this call.

-1 Indicates a QIO directive failure. The directive error code is placed in IOSB(1) in IBUF.

Note

The ind parameter is not supported by the K-series support routines. If compatibility with K-series support routines is desired, this parameter must be ignored.

12.3.1.2 CLOCKA—Set Clock A Rate

The CLOCKA routine sets the rate for Clock A. This routine is called by using the format shown next.

Format

CALL CLOCKA (irate,iprset,[ind],[lun])

Parameters

irate

Specifies the clock rate. One of the following must be specified:

-1 Direct-coupled Schmitt Trigger 1 (used only for A/D sweeps in dedicated mode +512; not supported by K-series support routines)

0 Clock B overflow or no rate

1 1 megahertz (MHz)

2 100 kilohertz (kHz)

3 10 kHz

4 1 kHz

5 100 hertz (Hz)

6 Schmitt Trigger 1

7 Line frequency

iprset

Specifies two's complement value for clock preset. The clock rate divided by the negative clock preset value yields the clock overflow rate. For example, to obtain a clock overflow rate of 10 kHz with a clock rate of 1 MHz, iprset equals -100 (minus 100_{10}). You can use the XRATE routine to calculate a clock preset value.

ind

Receives a success or failure code as follows:

0 Indicates an illegal argument list or I/O error. Possible causes are: microcode not loaded; driver not loaded; device off line or not in system.

1 Indicates Clock A set to start when sweep requested.

lun

Specifies the logical unit number (LUN). The default is 7.

12.3.1.3 CLOCKB—Control Clock B

The CLOCKB routine gives your task control over the KW11-K Clock B.

Format

CALL CLOCKB ([irate],iprset,mode,[ind],[lun])

Parameters

irate

Specifies the clock rate. When irate is nonzero, the clock is set running at the selected rate after the preset value specified by iprset is loaded. A 0 irate stops the clock. When irate is 0 or default, the iprset and mode parameters are ignored.

The following values are acceptable for irate:

0 Stop Clock B

1 1 MHz

2 100 kHz

3 10 kHz

4 1 kHz

5 100 Hz

6 Schmitt Trigger 3

7 Line frequency

iprset

Specifies the count by which to divide clock rate to yield overflow rate. You can use overflow events to drive Clock A. The preset parameter must be specified as 0 or as a negative number in the range -1 to -255. The value in iprset can be established by use of the XRATE routine.

mode

Specifies options. The following mode bit values listed that are preceded by a plus sign (+) are independent and can be added or ORed together. Those values not preceded by a plus sign are mutually exclusive and you can use only one such value at a time. All bit values not listed as follows are reserved:

1 Indicates Clock B operates in noninterrupt mode. The 16-bit clock is not incremented or altered. This allows a greater than 10-kHz pulse to be sent to Clock A.

+2 Indicates that the feed B-to-A bit is set in the Clock B status register.

ind

Receives a success or failure code as follows:

0 Indicates an illegal argument list or I/O error. Possible causes are: microcode not loaded; driver not loaded; or device off line or not in system.

1 Indicates Clock B started.

lun

The logical unit number (LUN). The default LUN is 7.

12.3.1.4 CVADF—Convert A/D Input to Floating Point

The CVADF routine converts an analog-to-digital (A/D) input value to a floating-point number. The routine can be invoked as a subroutine or a function by using the formats shown next.

Formats

```
CALL CVADF  (ival,val)

val  = CVADF(ival)
```

Parameters

ival

Specifies a value obtained from A/D input. Bits 12 to 15 are 0. Bits 0 to 11 represent the value.

val

Specifies that a real constant (REAL*4) receives the converted value. The converted value is calculated with the following formula:

```
val = (64*ival)/gain
```

12.3.1.5 DASWP—Initiate Synchronous D/A Sweep

The DASWP routine initiates synchronous digital-to-analog D/A output to an AA11-K.

Format

```
CALL DASWP  (ibuf,lbuf,[nbuf],[mode],[idwell],[iefn],ldelay,[ichn],[nchn],[ind])
```

Parameters

ibuf

Specifies a 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

Specifies the size, in words, of each data buffer. All data buffers must be equal in size and lbuf must be greater than 5. In dedicated mode, lbuf must be at least 257 words.

nbuf

Specifies the number of buffers to be emptied. If nbuf is omitted or set equal to 0, indefinite sweeping occurs. The STPSWP routine terminates indefinite sweeping.

mode

Specifies the start criteria. Except where noted, the plus sign (+) preceding mode bit values listed indicates that they are independent and can be added or ORed together. All bit values not listed are reserved.

The following values can be specified:

0	Indicates immediate start. This is the default.
+64	Multirequest mode. If this value is not specified, the request is for dedicated mode. If the request mode does not match the mode of the hardware (that is, different microcode in master microprocessor), the LPA11-K rejects the request with an appropriate error code.
+4096	Start method. If set, digital input start. If clear, immediate start. LAMSKS must be called to specify a digital start channel and a digital start mask (multirequest mode only).
+16384	Data overrun NONFATAL/FATAL. If selected, data overrun is considered nonfatal. The LPA11-K empties buffer 0. (See Section 12.4 for a discussion of buffer management.)

idwell

Specifies the number of clock overflows (pulses) between data sample sequences. For example, if idwell is 20 and nchn is 3, the following occurs: After 20 pulses, 1 channel is emptied on each of the next 3 pulses. Then, no emptying takes place for the next 20 pulses. In multirequest mode, this facility permits different rates for the same hardware clock rate and preset. In dedicated mode, the clock hardware rate controls sampling and idwell in the sweep start command is ignored.

If compatibility with K-series support routines is desired, your task must first establish the clock preset by calling the CLOCKA routine. You must use the default value (1) for the idwell parameter in the sweep start command. For the K-series, this procedure sets the rate as desired. For the LPA11-K, this procedure results in idwell in the sweep call defaulting to 1, thus yielding the same clock rate.

Note

This parameter is called iprset in the K-series support routines described in Chapter 13. Its function is different from the idwell parameter described here.

iefn

Specifies an event flag number (1 to 28, 30 to 96), or the name of a completion routine, or 0. If you use 0 or default iefn, the driver uses event flag 30 for internal synchronization. If you specify iefn as an event flag, the driver sets the event flag as each buffer is filled. Note that the LPA11-K support routines reserve event flag 29 for internal synchronization. If iefn is greater than 96, it is considered to be a completion routine that is called with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement).

FORTRAN completion routines must not contain any of the following:

- Any I/O through the FORTRAN run-time system
- Any use of virtual arrays
- Any use of floating-point operations
- Any errors, because error reporting is done through the FORTRAN run-time system
- Anything else that may change the FORTRAN impure area

Any of the above may result in fatal task errors or unpredictable results.

If multiple sweeps are initiated, your task should specify different event flags. This limitation cannot be enforced by the LPA11 driver.

ldelay

Specifies the delay from start event (DR11-K) until the first sample in irate units. A minimum delay of 150 microseconds is required (not verified by the LPA11 driver). This feature is supported in multirequest mode only.

ichn

Specifies the first channel number. The default is channel number 0.

nchn

Specifies the number of channels. The default is one channel.

ind

Receives a success or failure code as follows:

1 Indicates that the sweep was successfully initialized.

0 Indicates an illegal argument list, or SETIBF was not called prior to this call.

-1 Indicates a QIO directive failure. The directive error code is placed in IOSB(1) in IBUF.

Note

The ind parameter is not supported by the K-series support routines. If compatibility with K-series routines is desired, this parameter must be ignored.

12.3.1.6 DISWP—Initiate Synchronous Digital Input Sweep

The DISWP routine initiates a synchronous digital input sweep through a DR11-K. It can be called in multirequest mode only.

Format

CALL DISWP (ibuf,lbuf,[nbuf],[mode],[idwell],[iefn],[ldelay],[iunit],[nchn],[ind])

Parameters

ibuf

Specifies a 40-word array initialized by the SETIBF routine. The first two words of the array are the IOSB.

lbuf

Specifies the size in words of each data buffer. All data buffers must be equal in size and lbuf must be greater than 5.

nbuf

Specifies the number of buffers to be filled. If nbuf is 0 or is the default, indefinite sampling occurs. The STPSWP routine terminates indefinite sampling.

mode

Specifies the sampling options. The default is 0. The plus signs (+) preceding the mode bit values listed indicate that they are independent and can be added or ORed together. All bit values not listed as follows are reserved.

The following values can be specified:

0	Immediate start. This is the default mode.
+512	External trigger. The input sampling is triggered by interrupts generated by the DR11-K's external control lines, or its input bits if they are interrupt enabled.
+1024	Time-stamped sampling with Clock B. The doubleword consists of one data word followed by the value of the 16-bit clock at the time of the sample. IOSB(2) contains the number of 2-word samples in the buffer.
+2048	Event marking. LAMSKS must be called to specify an event mark word and an event mark mask.
+4096	Start method. If specified, digital input start. If clear, immediate start. LAMSKS must be called to specify a digital start channel and a digital start mask. The digital start channel need not differ from the input channel (iunit).
+16384	Data overrun NONFATAL/FATAL. If selected, data overrun is considered nonfatal. The LPA11-K fills buffer 0. (See Section 12.4 for a discussion of buffer management.)

dwell

Specifies the number of clock overflows (pulses) between data sample sequences. As an example, if idwell is 20 and nchn is 3, the following occurs: After 20 pulses, 1 channel is sampled on each of the next 3 pulses. Then, no sampling takes place for the next 20 pulses. In multirequest mode, this facility permits different sample rates for the same hardware clock rate and preset.

If compatibility with K-series support routines is desired, your task must first establish the clock preset by calling the CLOCKA routine. You must use the default value (1) for the idwell parameter in the sweep start command. For the K-series, this procedure sets the rate as desired. For the LPA11-K, this procedure results in idwell in the sweep call defaulting to 1, thus yielding the same clock rate.

Note

This parameter is called iprset in the K-series support routines described in Chapter 13. Its function is different from the idwell parameter described here.

iefn

Specifies an event flag number (1 to 28, 30 to 96), or the name of a completion routine, or 0. If you specify 0 or default this value, the driver uses event flag 30 for internal synchronization. If iefn is a valid event flag, the driver sets the selected event flag as each buffer is filled. Note that the LPA11-K support routines reserve event flag 29 for for internal synchronization. If iefn is greater than 96, it is considered to be a completion routine that is called with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement).

FORTRAN completion routines must not contain any of the following:

- Any I/O through the FORTRAN run-time system
- Any use of virtual arrays
- Any use of floating-point operations
- Any errors, because error reporting is done through the FORTRAN run-time system
- Anything else that may change the FORTRAN impure area

Any of the above may result in fatal task errors or unpredictable results.

If multiple sweeps are initiated, your task should specify different event flags. This limitation cannot be enforced by the LPA11 driver.

idelay

Specifies the delay from start event (DR11-K) until the first sample in irate units. The default or 0 indicates no delay.

iunit

Specifies the DR11-K unit number. The default is unit number 0. Values 0 to 4 are valid.

nchn

Specifies the number of channels. The LPA11-K treats each DR11-K in its configuration as one channel. The default is one channel.

ind

Receives a success or failure code as follows:

1 Indicates that the sweep was initialized successfully.

0 Indicates an illegal argument list, or SETIBF was not called prior to this call.

-1 Indicates a QIO directive failure. The directive error code is placed in IOSB(1) in IBUF.

Note

The nchn and ind parameters are not supported by the K-series support routines. If compatibility with K-series support routines is desired, these last two parameters must be ignored.

12.3.1.7 DOSWP—Initiate Synchronous Digital Output Sweep

The DOSWP routine initiates a synchronous digital output sweep through a DR11-K. It can be called in multirequest mode only.

Format

CALL DOSWP (ibuf,lbuf,[nbuf],[mode],[idwell],[iefn],ldelay,[iunit],[nchn],[ind])

Parameters

ibuf

Specifies a 40-word array initialized by the SETIBF routine. The first two words of the array are the IOSB.

lbuf

Specifies the size, in words, of each data buffer. All data buffers must be equal in size and lbuf must be greater than 5.

nbuf

Specifies the number of buffers to be emptied. If nbuf is 0 or is the default, indefinite emptying occurs. The STPSWP routine terminates indefinite emptying.

mode

Specifies the start criteria.

The following values can be specified in the high-order byte of mode:

0	Immediate start. This is the default mode.
+512	External trigger. The output sampling is triggered by interrupts generated by the DR11-K's external control lines or its input bits if they are interrupt enabled.
+4096	Start method. If set, digital input start. If clear, immediate start. LAMSKS must be called to specify a digital start channel and a digital start mask. The digital start channel need not differ from the output channel (iunit).
+16384	Data overrun NONFATAL/FATAL. If selected, data overrun is considered nonfatal. The LPA11-K fills buffer 0. (See Section 12.4 for a discussion of buffer management.)

idwell

Specifies the number of clock overflows (pulses) between data sample sequences. For example, if idwell is 20 and nchn is 3, the following occurs: After 20 pulses, 1 channel is activated on each of the next 3 pulses. Then, no output takes place for the next 20 pulses. In multirequest mode, this facility permits different output rates for the same hardware clock rate and preset.

If compatibility with K-series support routines is desired, your task must first establish the clock preset by calling the CLOCKA routine. You must use the default value (1) for the idwell parameter in the sweep start command. For the K-series, this procedure sets the rate as desired. For the LPA11-K, this procedure results in idwell in the sweep call defaulting to 1, thus yielding the same clock rate.

Note

This parameter is called iprset in the K-series support routines described in Chapter 13. Its function is different from the idwell parameter described here.

iefn

Specifies an event flag number (1 to 28, 30 to 96), or the name of a completion routine, or 0. If you specify 0 or default this value, the driver uses event flag 30 for internal synchronization. If iefn is a valid event flag, the driver sets the selected event flag as each buffer is emptied. Note that the LPA11-K support routines reserve event flag 29 for internal synchronization. If iefn is greater than 96, it is considered to be a completion routine that is called with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement).

FORTRAN completion routines must not contain any of the following:

- Any I/O through the FORTRAN run-time system
- Any use of virtual arrays
- Any use of floating-point operations
- Any errors, because error reporting is done through the FORTRAN run-time system
- Anything else that may change the FORTRAN impure area

Any of the above may result in fatal task errors or unpredictable results.

If multiple sweeps are initiated, your task should specify different event flags. This limitation cannot be enforced by the LPA11 driver.

idelay

Specifies the delay from start event (DR11-K) until the first sample in irate units. A minimum delay of 150 microseconds is required (not verified by the LPA11 driver).

iunit

Specifies the DR11-K unit number. The default is unit number 0. Values 0 to 4 are valid.

nchn

Specifies the number of channels. The LPA11-K treats each DR11-K in its configuration as one channel. Default is one channel.

ind

Receives a success or failure code as follows:

1 Indicates that the sweep was initiated successfully.

0 Indicates an illegal argument list or SETIBF was not called prior to this call.

-1 Indicates a QIO directive failure. The directive error code is placed in IOSB(1) in IBUF.

Note

The nchn and ind parameters are not supported by the K-series support routines. If compatibility with K-series support routines is desired, these last two parameters must be ignored.

12.3.1.8 FLT16—Convert Unsigned Integer to a Real Constant

The FLT16 routine converts an unsigned 16-bit integer to a real constant (REAL*4). It can be invoked as a subroutine or a function.

Formats

```
CALL FLT16   (ival,val)

val=FLT16(ival[,val])
```

Parameters

ival

Specifies an unsigned 16-bit integer.

val

Specifies the converted (REAL*4) value.

12.3.1.9 IBFSTS—Get Buffer Status

The IBFSTS routine returns information on buffers being used in a sweep.

Format

```
CALL IBFSTS   (ibuf,istat)
```

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

istat

Specifies an array with as many elements as there are buffers involved in the sweep. The maximum is eight. IBFSTS fills each element in the array with the status of the corresponding buffer. The possible status codes are as follows:

+2 Indicates that the buffer is in the device queue. That is, RLSBUF has been called for this buffer.

+1 Indicates that the buffer is in the task queue. That is, it is full of data (for input sweeps) or is available to be filled (for output sweeps).

0 Indicates that the status of the buffer is unknown. That is, it is not the current buffer nor is it in either the device or the user task queue.

-1 Indicates that the buffer is currently in use.

12.3.1.10 IGTBUF—Return Buffer Number

The IGTBUF routine returns the number of the next buffer to use. This routine should be called by your task's completion routines to determine which is the next buffer to access. Do not use it if an event flag was specified in the sweep-initiating call; if an event flag was specified, use the IWTBUF routine.

IGTBUF can be invoked as a subroutine or a function.

Formats

```
CALL IGTBUF   (ibuf,ibufno)


ibufno=IGTBUF(ibuf[,ibufno])
```

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

ibufno

Receives the number of the next buffer to access. If there is no buffer in the queue, ibufno contains -1.

On the return from a call to IGTBUF, the following are the possible combinations of ibufno and I/O status block (IOSB) contents:

ibufno	IOSB(1)	IOSB(2)	Explanation
n	400_8	(Word count)	Normal buffer complete.
n	1	(Word count)	Buffer complete. Sweep terminated. There may be additional buffers in the queue filled and ready for processing.
-1	0	0	No buffers in queue. Request still active.
-1	1	0	No buffers in queue. Sweep terminated.
-1	RSX-11M error code (decimal)	LPA11-K error code (octal)	No buffers in queue. Sweep terminated due to error condition. Note that the error is not returned until there are no more buffers in the task queue.

12.3.1.11 INXTBF—Set Next Buffer

The INXTBF routine alters the normal buffer selection algorithm. It allows your task to specify the number of the next buffer to be filled or emptied.

INXTBF can be invoked as a subroutine or a function.

Formats

CALL INXTBF (ibuf,ibufno[,ind])

ind = INXTBF(ibuf,ibufno[,ind])

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

ibufno

Specifies the number of the next buffer your task wants filled or emptied. The buffer must already be in the device queue.

ind

Receives an indication of the result of the operation as follows:

0 Indicates that the specified buffer was not in the device queue.

1 Indicates that the next buffer was set successfully.

12.3.1.12 IWTBUF—Wait for Buffer

The IWTBUF routine allows your task to wait for the next buffer to fill or empty. Use IWTBUF with the specification of an event flag in the sweep-initiating call. Do not use this routine if a completion routine was specified in the call to initiate a sweep; when event flags are specified, use the IGTBUF routine.

IWTBUF can be invoked as a subroutine or a function.

Formats

CALL IWTBUF (ibuf,[iefn],ibufno)

ibufno=IWTBUF(ibuf,[iefn],[ibufno])

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

iefn

Specifies the event flag on which the task waits. This should be the same event flag as specified in the sweep-initiating call. If you specify iefn as 0 or default this value, event flag 30 is used.

ibufno

Receives the number of the next buffer to be filled or emptied by your task.

On the return from a call to IWTBUF, the following are the possible combinations of ibufno and IOSB contents:

ibufno	IOSB(1)	IOSB(2)	Explanation
n	400_8	(Word count)	Normal buffer complete.
n	1	(Word count)	Buffer complete. Sweep terminated. There may be additional buffers in the queue filled and ready for processing.
-1	1	0	No buffers in queue. Sweep terminated.
-1	RSX–11M error code (decimal)	LPA11-K error code (octal)	No buffers in queue. Sweep terminated due to error condition. Note that the error is not returned until there are no more buffers in the task queue.

12.3.1.13 LAMSKS—Set Masks Buffer

The LAMSKS routine initializes a task buffer containing a LUN, a digital start mask and event mark mask, and channel numbers for the two masks. The routine then assigns the LUN. Each DR11-K is considered to be one channel. Each channel has both input and output capabilities.

LAMSKS must be called if digital input starting or event marking is to be used, or if a LUN other than the default LUN 7 is assigned to LA0. LAMSKS must also be called if your task uses multiple LPA11-Ks. If LAMSKS is to be called, it must be called prior to calling SETIBF. Unlike SETIBF, LAMSKS does not have to be called before each sweep initiation unless one or more parameters are to be changed.

Format

CALL LAMSKS (lamskb,[lun],[iunit],[idsc],[iemc],[idsw],[iemw],[ind])

Parameters

lamskb
Specifies a 40-word array.

lun
Specifies a logical unit number. The default LUN is 7.

iunit
Specifies the physical unit number of the LPA11-K. The default physical unit number is LA0.

idsc
Specifies the digital start word channel. The default is channel 0.

iemc
Specifies the event mark word channel. The default is channel 0.

idsw
Specifies the digital start word mask. The default is 0 (disable digital input starting).

iemw

Specifies the event mark word mask. The default is 0 (disable event marking).

ind

Receives a success or failure code as follows:

1 Indicates successful initialization.

0 Indicates an illegal argument list.

-n Indicates a LUN assignment failure; n is the directive error code.

Note

If compatibility with K-series support routines is desired, ignore this parameter.

For a discussion of event marking and digital starting, see the *LPA11-K Laboratory Peripheral Accelerator User's Guide.*

12.3.1.14 RLSBUF—Release Data Buffer

The RLSBUF routine declares one or more buffers free for use by the interrupt service routine.

The RLSBUF routine must be called to release a buffer or buffers to the device queue before the sweep is initiated. The device queue must always contain at least one buffer to maintain continuous sampling. Otherwise, buffer overrun occurs (see Section 12.4 for a discussion of buffer management). Note that RLSBUF does not verify whether the specified buffers are already in a queue.

Format

CALL RLSBUF (ibuf,[ind],n0[,n1...,n7])

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

ind

Receives a success or failure code as follows:

0 Indicates illegal buffer number specified, illegal number of buffers specified, or a double buffer overrun has been detected.

1 Indicates the buffer or buffers successfully released.

n0,n1,etc.

Specifies the numbers (0 to 7) of the buffers to be released. A maximum of eight can be specified.

12.3.1.15 RMVBUF—Remove Buffer from Device Queue

The RMVBUF routine removes a buffer from the device queue

Format

CALL RMVBUF (ibuf,n[,ind])

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

n

Specifies the number of the buffer to remove.

ind

Receives a success or failure code as follows:

0 Indicates that the specified buffer was not in the device queue.

1 Indicates that the specified buffer was removed from the queue.

12.3.1.16 SETADC—Set Channel Information

The SETADC routine establishes channel start and increment information for all sweeps. The SETIBF routine must be called to initialize the 40-word array (ibuf) before SETADC is called.

If, in the call to SETADC, nchn is 1 or inc is 0, the single channel bit is set in the mode word of the start Request Descriptor Array (RDA) when the sweep start routine is called.

SETADC can be invoked as a subroutine or a function.

Formats

CALL SETADC (ibuf,[iflag],[ichn],[nchn],[inc],[ind])

ind = ISTADC(ibuf,[iflag],[ichn],[nchn],[inc],[ind])

Parameters

ibuf

Specifies a 40-word array initialized by the SETIBF routine.

iflag

Ignored. It is included for compatibility with K-series support routines.

ichn

Specifies the first channel number. The default is 0. If inc equals 0 (or default), ichn is the address of a random channel list. A random channel list is an array of n elements, where each element is a channel number. The final element must have bit 15 set to indicate the end of the list.

nchn

Specifies the number of samples to be taken per sequence. The default is one sample.

inc

Specifies the channel increment. The default is 1. You should specify an increment of 2 for differential A/D input. If inc equals 0, ichn is an array of random channels to receive input.

ind

Receives a success or failure code as follows:

0 Indicates an illegal channel number or SETIBF was not called prior to the SETADC call.

1 Indicates successful recording of channel information for the sweep call.

12.3.1.17 SETIBF—Set Array for Buffered Sweep

The SETIBF routine initializes an array required by buffered sweep routines. The SETIBF routine must be called before every call to a buffered sweep routine.

Format

CALL SETIBF (ibuf,[ind],[lamskb],buf0[,buf1...buf7])

Parameters

ibuf

Specifies a 40-word array.

ind

Receives a success or failure code as follows:

0 Indicates a parameter or buffer error.

1 Indicates the array was initialized successfully.

lamskb

Specifies the name of a 40-word array. This array allows the use of multiple LPA11-Ks within the same program because the LUN is specified in the first word of the array. Refer to the description of the LAMSKS routine.

If you want compatibility with K-series software, use the default (LUN 7) lamskb parameter, and LUN 7 is assigned to LA0 in the task-build command file for your task.

buf0...buf7

Specifies the name of a buffer. A maximum of eight buffers can be specified. Any buffer names in excess of eight are ignored. At least two buffers must be specified to maintain continuous sampling.

Each buffer specified in the call to SETIBF is assigned a number ranging from 0 to 7.

The assignment of these numbers is based on the order in which buffer names appear in the argument list. The first buffer whose name appears in the list is assigned number 0, the second is assigned number 1, and so forth. In all subsequent calls to other routines involving the set of buffers specified in a call to SETIBF, these numbers, rather than names, refer to particular buffers.

12.3.1.18 STPSWP—Stop Sweep

The STPSWP routine stops a sweep that is in progress.

Format

CALL STPSWP (ibuf[,iwhen],[ind])

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

iwhen

Specifies when to stop the sweep as follows:

0 Stops the sweep and immediately aborts the sweep. This is the default stop method. The sweep is stopped asynchronously by the LPA11-K hardware. When IOSB(1) equals 1, the sweep has been stopped. Call IWTBUF continuously after calling STPSWP until the sweep has actually been stopped. When stopping (aborting) a sweep in this manner, the data contents of the current data buffer cannot be guaranteed.

+n (Any positive value) Stops the sweep at the end of the current buffer. This is considered to be the normal means for stopping a sweep.

-n (Any negative value) Reserved. (Do not use.)

ind

Receives a success or failure code as follows:

1 Indicates that the sweep is stopped (at the time indicated by iwhen).

0 Indicates an illegal argument list.

-n Is a directive error code indicating that the stop sweep QIO failed.

12.3.1.19 XRATE—Compute Clock Rate and Preset

The XRATE routine allows your task to compute a clock rate and preset. The clock rate divided by the clock preset yields the desired dwell (intersample interval).

Note

You can use the XRATE routine only on systems that have a FORTRAN or BASIC-PLUS-2 compiler. This module is not included with the other LPA11-K support routines in object module format. Rather, it is included in source code format with the K-series source modules in directory [45,10] on the system disk.

XRATE can be invoked as a subroutine or a function.

Formats

CALL XRATE (dwell,irate,iprset,iflag)

adwell = XRATE(dwell,irate,iprset,iflag)

Parameters

dwell

Specifies the intersample time desired by your task. The time is expressed in decimal seconds (REAL*4).

irate

Receives the computed clock rate as a value from 1 to 5.

iprset

Receives the clock preset.

iflag

Specifies whether the computation is for Clock A or Clock B as follows:

0	Indicates the computation is for Clock A.
Nonzero	Indicates the computation is for Clock B.

adwell

Specifies the actual dwell rate for the clock based on the irate and iprset parameters.

12.3.2 MACRO–11 Interface

The MACRO–11 interface to the LPA11-K consists of either the callable routines described in Section 12.3.1 or a set of device-specific QIO functions.

12.3.2.1 Accessing Callable LPA11-K Support Routines

MACRO–11 programmers access the LPA11-K support routines through either of the following two techniques:

- The standard subroutine linkage mechanism and the CALL operation code
- Special-purpose macros that generate an argument list and invoke a subroutine

These techniques are described in the following subsections.

12.3.2.2 Standard Subroutine Linkage and CALL Op Code

LPA11-K routines can be accessed through use of the standard subroutine linkage mechanism and the CALL operation code. The format of this procedure is as follows:

```
          .PSECT  code
          MOV     #arglist,R5     ;ARGUMENT ADDRESS TO R5
          CALL    lsubr           ;CALL LPA11-K ROUTINE

          .PSECT  data
arglist:  .BYTE   narg,0          ;NUMBER OF ARGUMENTS
          .WORD   addr1           ;FIRST ARGUMENT ADDRESS
    .
    .
    .
          .WORD   addrn           ;LAST ARGUMENT ADDRESS
```

In this sample, the two program section (PSECT) directives are shown only to indicate the noncontiguity of the code and data portions of the linkage mechanism. Within the argument list, any argument that is to be defaulted must be represented by a -1 address (that is, 177777_8).

12.3.2.3 Special-Purpose Macros

To facilitate the calling of LPA11-K support routines from a MACRO–11 program, two macros are provided in file [45,10]LABMAC.MAC. These macros are as follows:

- INITS
- CALLS

INITS is an initialization macro. It must be invoked at the beginning of the MACRO–11 source module.

CALLS invokes an LPA11-K support routine.

Format

CALLS lsubr, <arg1,...,argn>

Parameters

lsubr

Specifies the name of an LPA11-K support routine.

arg1,...,argn

Specifies the arguments to be formatted into an argument list and to be passed to the routine. Each argument can be either a symbolic name or a constant (interpreted as a positive decimal number), or it can be defaulted.

Example

```
        .TITLE  EXAMPLE
        .IDENT  /01.00/
IBUF:   .BLKW   40.
ISTAT:  .BLKW   5
        INITS                   ; INITIALIZATION
    .
    .
    .
START:
    .
    .
    .
```

```
;
; FIND STATUS OF 5 SWEEP BUFFERS
; USED IN THE CURRENT SWEEP
;
        CALLS   IBFSTS(IBUF,ISTAT)
    .
    .
    .
        .END    START
```

Illustrates the use of special-purpose macros.

12.3.2.4 Device-Specific QIO Functions

Table 12–2 lists the device-specific functions of the QIO system directive macro that are available for the LPA11-K. Programmers using these functions are entirely responsible for buffer management (refer to Section 12.4) as well as all other interfaces (for example, the request descriptor array). Little (if any) performance improvement over the use of FORTRAN support routines can be expected by using QIOs. Therefore, you should use the routines described in Section 12.3.1.

Table 12–2: Device-Specific QIO Functions for the LPA11-K

QIO Function	Purpose
IO.CLK	Start clock
IO.INI	Initialize LPA11-K
IO.LOD	Load microcode
IO.STA	Start data transfer
IO.STP	Stop request

The MACRO–11 programmer must set up the appropriate Request Descriptor Array (RDA) before the corresponding QIO request is issued. In the case of the IO.STA function (start data transfer), the RDA is set up with buffer virtual addresses. The LPA11-K driver address checks and relocates these buffers, changing them from single-word to doubleword addresses. The RDA is fully described in the source code of the driver.

IO.CLK—Start Clock

The IO.CLK function writes an image into the LPA11-K real-time clock control register and issues a clock start command.

Format

```
QIO$C   IO.CLK,..., <mode,ckcsr,preset>
```

Parameters

mode
Specifies the mode.

ckcsr
Specifies the image to be written into the clock control register. To achieve the function of clock rate -1 (see Section 12.3.1.2) for Clock A only, set a clock rate of 0 and set the Schmitt Trigger 1 Interrupt Enable bit in the Clock A Status Register.

preset
Specifies the clock preset.

IO.INI—Initialize the LPA11-K

The IO.INI function initializes the LPA11-K. The task issuing the QIO request must be privileged.

Format

QIO$C IO.INI,..., <irbuf,278.>

Parameter

irbuf
Specifies a buffer containing an LPA11-K that initializes RDA. The buffer size must be at least 278_{10} bytes.

IO.LOD—Load Microcode

The IO.LOD function loads a buffer of LPA11-K microcode. The issuing task must be privileged. The function verifies that there are no active tasks for the LPA11-K and resets the hardware. It then loads and verifies the microcode, starts the LPA11-K, and enables interrupts. The function returns to the issuing task when the Ready Interrupt message is posted.

Format

QIO$C IO.LOD,..., <mbuf,2048.>

Parameter

mbuf
Specifies a buffer containing microcode to be loaded. The buffer size must be 2048_{10} bytes.

IO.STA—Start Data Transfer

The IO.STA function issues an LPA11-K data transfer start command.

Format

QIO$C IO.STA,..., <bufptr,40.>

Parameter

bufptr

Specifies a pointer to a buffer containing an LPA11-K sample start RDA. The buffer size must be at least 40_{10} bytes.

The subfunction codes defined for the IO.STA function are as follows:

Bit 0 = 0 Indicates that an asynchronous system trap (AST) is to be generated for every buffer (if an AST is specified).

Bit 0 = 1 Indicates that an AST is to be generated only for exception conditions.

IO.STP—Stop Request

The IO.STP function stops a data transfer request. The issuing task must be the same task that initiated the data transfer.

Format

QIO$C IO.STP,..., <userid>

Parameter

userid

Specifies the index number associated with the task whose request is to be stopped.

12.3.3 The I/O Status Block

Each active sweep must have its own I/O status block (IOSB). The IOSB is a 2-word array allocated in your task. Use it to receive the status of a call to an LPA11-K support routine. When your task calls a data sweep routine, the IOSB is always the first two words of the 40-word array specified as the first argument of the call. The first word of the IOSB contains the status code, and the second word contains the buffer size in words.

Note

The LPA11-K driver does not directly use the 2-word IOSB. Instead, the driver uses a 4-word IOSB for internal communications with support routines; this 4-word IOSB is completely transparent to those tasks that use FORTRAN support routines. However, when issuing QIOs, it is the 4-word IOSB that must be referenced.

The first two words of the 4-word IOSB function as a 2-word overall IOSB for returning QIO completion status. The driver returns status such as sweep done, system errors, and LPA11-K hardware errors with this 2-word portion of the IOSB.

The remaining two words function as an intermediate IOSB for passing status information during the data sweeps. MACRO–11 programs using QIO calls always receive the correct 2-word portion of the IOSB in the AST generated by the LPA11-K driver.

The codes that can appear in the first word of an IOSB are in ISA-compatible format (with the exception of the I/O pending condition). Table 12–3 lists all return codes (except 351; see Section 12.5).

Table 12–3: Contents of First Word of IOSB

IOSB(1)		
FORTRAN	**MACRO**	**Meaning**
0	IO.PND	Operation pending; I/O in progress
1	IS.SUC	Successful completion
301	IE.BAD	Invalid arguments
302	IE.IFC	Invalid function code
303	IE.DNR	Device not ready (See Section 12.7)
304	IE.VER	Unrecoverable hardware error caused by powerfail
305	IE.ULN	LUN not assigned to LPA11-K
306	IE.SPC	Illegal buffer specification
309	IE.DUN	Insufficient UNIBUS Mapping Registers (UMRs) available for request
313[1]	IE.DAO	Data overrun
315[1]	IE.ABO	Request terminated; LPA11-K status code in IOSB(2)
316	IE.PRI	Privilege violation
317[1]	IE.RSU	Resource in use (load microcode only)
320	IE.BLK	Executive blocked driver waiting for UMRs
323	IE.NOD	System dynamic memory exhausted
359[1]	IE.FHE	Fatal hardware error on device
366	IE.BCC	LPA11-K load microcode error
397	IE.IEF	Invalid event flag specified

[1] IOSB(2) contains an LPA11-K status code. Refer to the *LPA11-K User's Manual* for explanation of status code.

12.4 Buffer Management

The management of buffers for data transfers by LPA11-K support routines involves the use of two FIFO (first-in/first-out) queues:

- The device queue (DVQ)
- The user task queue (USQ)

The device queue (DVQ) contains the numbers of all buffers that your task has released to the support routines in a call to RLSBUF. The buffers represented by these numbers are ready to be filled with data (input sweeps) or to be emptied of data (output sweeps). Any buffer specified in a call to INXTBF must already be in DVQ.

Your task queue (USQ) contains the numbers of buffers available to your task. For output sweeps, this queue contains the numbers of buffers that have already been emptied by the driver. For input sweeps, the buffers represented by USQ are those that are filled with data. In both instances, your task determines the next buffer to use (that is, it extracts the first element of USQ) by calling IGTBUF or IWTBUF.

Both the DVQ and USQ are initialized to -1—indicating no buffers—when your task calls the SETIBF routine. Your task must call RLSBUF before initiating any sweep because at least one buffer must be present in DVQ for the first input or output to occur.

For input sweeps, your task should call RLSBUF and specify the numbers associated with all the buffers to be used in the sweep.

For output sweeps, your task can specify two buffers (for continuous sweeps) in the call to RLSBUF. The first action then taken either in a completion routine or after a call to IWTBUF is to release the next buffer. However, note that this approach does not represent true multiple buffering because data overrun occurs if the second buffer is not released in time.

If a buffer overrun occurs, the LPA11-K normally aborts the affected sweep and returns an appropriate error code. However, the option of having buffer overruns treated as nonfatal error conditions can be selected by specifying the appropriate mode argument in any of the sweep calls. Then, when a buffer overrun occurs, the LPA11-K defaults to buffer 0 for its next data buffer. In this case, the following special considerations regarding buffer management must be observed.

Call RLSBUF before calling any of the sweep control calls. However, if buffer overruns are to be treated as nonfatal conditions, the task should not specify buffer 0 in the initial call to RLSBUF. (It is assumed at the outset that buffer 0 is available for use in this manner and, therefore, should not be released.)

Once a buffer overrun has occurred, the LPA11-K uses buffer 0 and places it on the task's queue just like any other data buffer. At this point, buffer 0 is no longer available for buffer overruns. The task then removes buffer 0 from the task queue by IWTBUF or IGTBUF for possible processing. It is the task's responsibility to release buffer 0 for future buffer overruns by specifying buffer 0 in a call to RLSBUF. Note that the task cannot determine that buffer overrun occurred until it receives buffer 0 from IWTBUF or IGTBUF.

The LPA11-K always uses buffer 0 following a buffer overrun if that condition was specified as nonfatal. Thus, when a second buffer overrun occurs before buffer 0 has been processed and made available for that purpose, a condition called "double buffer overrun" occurs. In this case, buffer 0 is not put on the task queue because the actual contents of buffer 0 cannot be determined at this time, and buffer 0 may actually still be on that queue. The double buffer overrun condition is detected when the task attempts to make buffer 0 available for future buffer overruns with the call to RLSBUF. Note that this is the first time that the task is notified of the condition. If a double buffer overrun condition is detected during the call to RLSBUF, the task must be notified of the condition indicating that the previous processing of buffer 0 contents may have been of no value (the LPA11-K probably changed the buffer's contents while it was being processed).

12.5 Loading the LPA-11 Microcode

LAINIT is a privileged task that loads all versions of LPA11-K microcode. When called, LAINIT issues an IO.LOD function in a QIO request, followed by IO.INI and IO.CLK function requests. The IO.CLK function starts the clock with a default clock rate of 1 megahertz (MHz).

During system generation, Phase 1, a command file is generated with LPA11-K support selected through operator response to system generation questions. During system generation, Phase 2, the command file builds LAINIT by using additional information obtained through operator response to system generation questions. This information further defines the LPA11-K's system environment and characteristics for your specific application.

Separate tasks are built during system generation that invoke LAINIT to load appropriate LPA11-K microcode. These tasks are named LAINn, where n corresponds to unit number (starting with unit number 0) for each LPA11-K unit in the system. Thus, you never directly invoke LAINIT.

System generation generates command lines in SYSVMR.CMD that install LAINIT and LAIN0; LAIN1 and subsequent LPA11-K unit-numbered tasks are not included in the command file. Thus, you must install these tasks (if they are required) with the Virtual Monitor Console Routine (VMR) or the Monitor Console Routine (MCR).

Once LAINIT and LAINn tasks have been installed, a particular version of LPA11-K microcode for a specific unit can be loaded by running the corresponding LAINn task. For example, the following command line executes LAIN2, loading microcode for LPA11-K unit 2:

```
>RUN LAIN2 [RET]
```

When a powerfail recovery occurs, the LPA11-K driver terminates all outstanding activity and requests execution of initiating the task or tasks (LAINn) for each unit. This provides powerfail recovery for the LPA11-K microprocessor, provided the LAINIT and LAINn tasks are installed. Note that when either the RSX–11M system is bootstrapped or the LPA11-K driver is loaded, a simulated powerfail (resulting in driver powerfail recovery) occurs, loading microcode for each LPA11-K unit. In addition, when the LPA11-K is brought on line on an RSX–11M–PLUS system, a simulated powerfail occurs.

If the request for the initiating task (LAINn) fails or the loader fails to load the driver, the LPA11-K unit does not become initialized. Any further attempt to use the LPA11-K fails, with the "Device not ready" (IE.DNR) error code returned to the requesting task.

If there is no LPA-11K present at the default address, LAINx returns error code 351 in IOSB(1). This failure occurs if there is more than one LPA-11K and the one at the default address is removed. There must always be an LPA-11K at the default address.

All versions of LAINn set the real-time clock frequency to 1 MHz by default. The Unit Control Block (UCB) device characteristics word 4 (U.CW4) contains a 16-bit buffer preset value that controls the rate of ticks (that is, the rate at which the clock interrupts). This value can be set dynamically or during system generation. The quotient resulting when this value is divided into 1 MHz is the rate of ticks. For example, if U.CW4 contains the value 2, the tick rate is 500 kilohertz (kHz). Your task can issue a Get LUN Information system directive to examine the preset value and the MCR command SET /BUF can modify the value while the system is running. This modification takes effect the next time the LPA11-K is reloaded with microcode by LAINx.

12.6 Unloading the Driver

To attain maximum LPA11-K performance, the LPA11-K driver appears idle to the RSX–11M–PLUS Executive. As a result, the potential problem exists that any privileged user can unload the driver while the LPA11-K is servicing other users. Therefore, the privileged user must first determine that the LPA11-K is not being used before he or she unloads the driver.

12.7 Timeout of the LPA11-K

The error code IO.DNR means that the LPA11-K timed out while processing your task request. In dedicated mode, this condition can have special meaning.

The LPA11-K driver (LADRV) disables the timeout countdown following LPA11-K acknowledgment of your task request. In all cases in multirequest mode, and in most cases in dedicated mode, this acknowledgment is received almost immediately after your task request is passed to the LPA11-K. The only case when this is not true is when your task requests that a data sweep be started while in dedicated mode. In this case, the LPA11-K waits to transfer the first 256 words of data before acknowledging the sweep request.

If a task is sampling at extremely slow data rates in dedicated mode, the time to transfer the first 256 words may exceed the timeout count for the device. This can be avoided by using the multirequest mode.

If a task must use dedicated mode for high sampling rates, and the start of the sweep is delayed for an extended period of time, the timeout count for the LPA11-K must be disabled. (Refer to the note in LADRV describing this timeout problem and showing where the timeout can be safely disabled for sweep calls.)

Note

This procedure disables the detection of real timeouts for sweep calls in dedicated mode.

12.8 22-Bit Addressing Support

The LPA11-K driver supports 22-bit addressing on systems that have capability. When the system employs 22-bit addressing, certain restrictions are imposed. As a result, tasks written for use with earlier LPA11-K driver versions may not run without modifying your task. These restrictions are discussed in the remainder of this section.

When the LPA11-K driver is executed on 22-bit systems, a certain contiguity of your task's data structures must be established. The task data transfer buffers and the IBUF array must be contiguous. In addition, the task random channel list (if present) and the last data transfer buffer must be contiguous. Thus, the correct sequence for your task data is the IBUF array, followed by the task data transfer buffers, followed by the task random channel list. Failure to structure your task's data in this manner can result in illegal buffer specification errors (IE.SPC) being returned or possible corruption of task address space by data sweeps.

Because the LPA11-K driver can potentially request more buffer space than there is UMR mapping space, a limit must be specified on the total number of UMRs that the LPA11-K driver can use at any time. You specify this limit during system generation, part 1, along with the interrupt vector and control and status register (CSR) address for the LPA11-K.

If a task's UMR requirements cause the total number of UMRs currently in use by the LPA11-K to exceed the limit specified during system generation, the task receives an "Insufficient UMRs available for request" (IE.DUN) error code in IOSB(1) of the IBUF array.

This condition can be avoided by setting the UMR limit to the expected minimum number required for smooth LPA11-K operation for all expected tasks. Because each UMR maps 8 kilobytes (Kb), each task's requirements can be calculated as follows:

1. Each IBUF array requires 76_{10} bytes of UMR mapping.
2. Add this result to the byte length of all the contiguous transfer buffers to be used in the sweep.
3. Add this result to the byte length of the random channel list (if it exists).
4. The number of UMRs your task needs is the total byte count divided by 8192 (8K) and rounded up to the next 8K (if not an exact multiple of 8192).

Because there are only 31 UMRs available for the entire system, it is not desirable to allow the LPA11-K driver (through the limit specified during system generation) to have access to all or nearly all UMRs at any given time. Because other device drivers may also require UMR mapping, the total allocation of UMRs by LADRV can slowly choke a system, and, for that reason, allocation of UMRs must be carefully considered.

The UMR allocation limit for the LPA11-K can be changed by directly modifying the value in the LPA11-K's UCB word U.LAUB; it is not necessary to do another system generation. Use the OPEN command to access and change the limit to the new value. Possible values can range from 0 to 31. Then, make the required change. UNLOAD and then LOAD the LPA11-K driver. If the LPA11-K driver is resident, the value in U.LAUB+2 must also be changed to the new value.

Note

Be sure the LPA11-K is idle before attempting to access the UCB.

It is possible for a condition to exist where there may not be enough UMRs available for the Executive to allocate to the driver at the time the request is made, even if the number of UMRs necessary to map your task's request are within the limit specified during system generation. When this happens, the Executive blocks the driver until its UMR request can be granted. Because this condition can introduce sweep timing errors, the current sweep is unconditionally aborted and an appropriate error code (IE.BLK) is returned to the task in IOSB(1).

12.9 Sample Programs

The following sample program shows the basic flow for programming the LPA11-K in a high level language.

```
C       LPA11-K SAMPLE PROGRAM
C
C SAMPLE SHOWS THE BASIC FLOW FOR PROGRAMMING THE LPA11-K IN A HIGHER
C LEVEL LANGUAGE.  IT IS EXPECTED THAT YOUR TASK TESTS IOSB RETURNS AND
C ERROR INDICATORS (IND) AS NECESSARY.  SYNCHRONOUS PROGRAM TERMINATION
C IS SUGGESTED. NOTE: THIS SAMPLE PROGRAM DOES NOT EXECUTE CORRECTLY IN
C 22-BIT MODE
C
C       D/A DEDICATED MODE WITH CONTINUOUS SAMPLING
C
C       PROGRAM RUNS 3 LOOPS (BASED ON NCNT).  ON FIRST LOOP,
C       STOPS SYNCHRONOUSLY AT END OF PRESENT BUFFER WHICH HAPPENS
C       TO BE BUFFER #3 BEING FILLED FOR THE 2ND TIME.
C       THE 2ND LOOP TERMINATES ASYNCHRONOUSLY (IWHEN=0).
C       THE 3RD LOOP TERMINATES ASYNCHRONOUSLY ALSO.
C
C
        DIMENSION IBUF(40),IOSB(2),NB(1024,8)
        EQUIVALENCE (IBUF(1),IOSB(1))
        EQUIVALENCE (N0,NB(1,1)),(N1,NB(1,2)),(N2,NB(1,3)),(N3,NB(1,4))
        EQUIVALENCE (N4,NB(1,5)),(N5,NB(1,6)),(N6,NB(1,7)),(N7,NB(1,8))
        CALL CLOCKA (4,-1)
        IWHEN=1
        NCNT=0
2       ICNT=1
    5   CALL SETIBF(IBUF,IND,,N0,N1,N2,N3)
C
C INITIALIZE BUFFERS TO ALL -2'S
C
        DO 10 J=1,8
        DO 10 K=1,1024
   10   NB(K,J)=-2
        CALL RLSBUF(IBUF,IND,1,2,3)
        CALL DASWP(IBUF,1024,,,,20)
20      CALL IWTBUF(IBUF,20,IBUFNO)
        CALL RLSBUF (IBUF,IND,IBUFNO)
        WRITE (1,300) IBUFNO,IOSB(1),IOSB(2),ICNT
        IF (NCNT.EQ.3) GOTO 40
        IF (ICNT.EQ.6) GOTO 2
        ICNT=ICNT+1
        IF (ICNT.NE.4) GOTO 20
        CALL STPSWP (IBUF,IBUFNO)
        IWHEN=0
        NCNT=NCNT+1
        GOTO 20
40      CALL IGTBUF(IBUF,IBUFNO)
        WRITE (1,300) IBUFNO,IOSB(1),IOSB(2),ICNT
300     FORMAT (3X,I10,2O8,I10)
        STOP
        END
```

The following sample program tests the digital I/O interface of the LPA11-K. It will execute correctly in 22-bit mode.

```
C
C PROGRAM TO TEST DIGITAL INPUT AND OUTPUT FOR LPA11-K
C DIGITAL EQUIPMENT CORPORATION
C
C THIS PROGRAM OUTPUTS A DATA BUFFER TO THE LPA11-K
C DIGITAL I/O INTERFACE AND AT THE SAME INSTANT, FOR EACH SAMPLE
C WORD, READS THE RESULTS BACK. THE DATA BUFFERS ARE COMPARED TO
C MAKE SURE THE TRANSFER IS COMPLETED SUCCESSFULLY.
C

C  ****** NOTE! ******
C       THIS PROGRAM WORKS IF AND ONLY IF THE DIGITAL I/O MODULE
C       UNIT SPECIFIED HAS THE MAINTENANCE JUMPER "WRAP-AROUND" CABLE
C       INSTALLED !!!!
C
C
C RESERVE STORAGE FOR LPA11-K ROUTINES
C

C       THIS PROGRAM WORKS IN 22-BIT MODE
C
C DATA BUFFERS
C
        INTEGER*2 IBUFI(40),INBUF(300,4)
        INTEGER*2 COMMI(1240)
        EQUIVALENCE(IBUFI(1),COMMI(1))
        EQUIVALENCE(INBUF(1,1),COMMI(41))

        INTEGER*2 IBUFO(40),OUTBUF(300,4)
        INTEGER*2 COMMO(1240)
        EQUIVALENCE(IBUFO(1),COMMO(1))
        EQUIVALENCE(OUTBUF(1,1),COMMO(41))

C RESERVE STORAGE AND EQUIVALENCE FOR RSX I/O STATUS BLOCKS
        LOGICAL*1 INIOS(4),OUTIOS(4)
        EQUIVALENCE (IBUFO(1),OUTIOS(1)),(IBUFI(1),INIOS(1))
C
C SET BUFFER SIZE TO USE FOR THIS REQUEST - MAXIMUM OF 300 WITHOUT
C CHANGING THE DIMENSION STATEMENTS. MUST BE EVEN!
        ISIZE=300

C
C INITIALIZE THE PASS COUNTER FOR THE LOOP
        IPASS=1
C
C SET LPA11-K LOGICAL UNIT NUMBER AND ASSIGN IT TO LA0:
        ILUN=7
        CALL ASSIGN(ILUN,'LA:',0,ISTAT)
        IF(ISTAT .LT. 0)GO TO 100
C
C INITIALIZE THE OUTPUT DATA BUFFER
        DO 2 J=1,4
        DO 2 I=1,ISIZE,2
        OUTBUF(I,J)="125252
        OUTBUF(I+1,J)="052525
2       CONTINUE
```

```
C
C STOP LPA11-K REAL TIME CLOCK "A" THIS ENSURES THAT
C NOTHING HAPPENS WHEN WE INITIALIZE THE TWO SWEEPS.
5       CALL CLOCKA(0,0,ISTAT,ILUN)
        IF(ISTAT .NE. 1)GO TO 110
C
C INITIALIZE THE INPUT DATA BUFFER. ASSUME THE LPA11-K DIGITAL
C I/O INTERFACE IS CONFIGURED IN THE DATA LATCH MODE (AS OPPOSED
C TO SENSE). THUS THE OUTPUT DATA BUFFER MUST CONTAIN A BIT CHANGE
C FOR EVERY BIT POSITION IN SUCCEEDING DATA WORDS.
        DO 10 J=1,4
        DO 10 I=1,ISIZE
        INBUF(I,J)=0
10      CONTINUE
C
C INITIALIZE DIGITAL OUTPUT SWEEP.  THIS MUST BE DONE BEFORE INIT
C OF DIGITAL INPUT SWEEP! THE LPA11-K PROCESSES THE TRANSFER OF
C DATA IN THE ORDER OF THE SPECIFICATION OF THE SWEEPS.  THUS WE
C WANT TO OUTPUT BEFORE WE INPUT.
        CALL SETIBF(IBUFO,ISTAT,,OUTBUF(1,1),OUTBUF(1,2),OUTBUF(1,3),
        1   OUTBUF(1,4))
        IF(ISTAT .NE. 1)GO TO 120
C
C RELEASE BUFFER FOR OUTPUT SWEEP
C ALL FOUR BUFFERS -- INDEXES 0,1,2,3 -- ARE RELEASED
        CALL RLSBUF(IBUFO,ISTAT,0,1,2,3)
        IF(ISTAT .NE. 1)GO TO 130
C
C "START" DIGITAL OUTPUT SWEEP. REMEMBER NOTHING HAPPENS UNTIL
C THE REAL TIME CLOCK STARTS. THE LPA11-K PROCESSES THE REQUEST
C AND IS ALREADY TO TRANSFER DATA WHEN WE RESUME THE CLOCK.
C EVENT FLAG 14 IS SPECIFIED. A DIFFERENT EVENT FLAG MUST BE
C SPECIFIED FOR THE DIGITAL INPUT SWEEP SO THE FORTRAN PROGRAM
C CAN SYNCHRONIZE WITH TWO INDEPENDENT, ASYNCHRONOUS PROCESSES.
        CALL DOSWP(IBUFO,ISIZE,4,0,1,14,30,0)
C
C
C NOW INITIALIZE FOR DIGITAL INPUT SWEEP.  THE SAMPLING PARAMETERS
C MUST BE THE SAME FOR BOTH THE INPUT AND OUTPUT SWEEP.  WE WANT
C TO WRITE AND READ THE SAME DATA WORD AT THE SAME TIME.
        CALL SETIBF(IBUFI,ISTAT,,INBUF(1,1),INBUF(1,2),INBUF(1,3),
        1   INBUF(1,4))
        IF(ISTAT .NE. 1)GO TO 140
C
C RELEASE THE INPUT BUFFERS
        CALL RLSBUF(IBUFI,ISTAT,0,1,2,3)
        IF(ISTAT .NE. 1)GO TO 150
C
C "START DIGITAL OUTPUT SWEEP. AGAIN, NOTHING HAPPENS UNTIL
C WE RESUME THE LPA11-K REAL TIME CLOCK.
C EVENT FLAG 15 IS SPECIFIED TO SEPARATE THE INPUT AND OUTPUT SWEEPS.
        CALL DISWP(IBUFI,ISIZE,4,0,1,15,30,0)
```

```
C
C NOW FOR THE BIG EVENT! WE START THE CLOCK AND SEE WHAT HAPPENS.
        CALL CLOCKA(1,-150,ISTAT,ILUN)
        IF(ISTAT .NE. 1)GO TO 150

C
C
C THE LPA11-K SHOULD NOW BEGIN TO TRANSFER DATA
C FIRST WE WAIT FOR THE DIGITAL OUTPUT SWEEP TO FINISH.  IT WAS
C STARTED FIRST AND SHOULD FINISH FIRST.  WE VERIFY THAT IT
C FINISHES CORRECTLY OR CHECK FOR ERRORS.
15      CALL IWTBUF(IBUFO,14,IBUFNO)
C
C IF BUFFER NUMBER IS -1, THEN ERROR
C IF BUFFER NUMBER IS 0,1, OR 2, THEN CONTINUE
C IF BUFFER NUMBER IS 3, THEN FINISHED
        IF(IBUFNO .LT. 0) GO TO 160

C
C NOW WAIT FOR THE DIGITAL INPUT SWEEP TO FINISH.  THE SAME ERROR
C CONDITIONS APPLY.
        CALL IWTBUF(IBUFI,15,IBUFNO)
        IF(IBUFNO .LT. 0)GO TO 170
        IF(IBUFNO .LE. 2)GO TO 15

C
C THE FACT THAT WE HAVE GOTTEN HERE SAYS THE LPA11-K HAS DONE ITS
C THING.
C CHECK THE INPUT DATA BUFFERS AGAINST THE OUTPUT DATA BUFFERS
        DO 20 J=1,4
        DO 20 I=1,ISIZE
        IF(INBUF(I,J) .NE. OUTBUF(I,J))GO TO 180
20      CONTINUE

C
C SUCCESSFUL COMPLETION, LET EVERYONE KNOW. THEN GO BACK AND DO IT
C AGAIN.
        WRITE(5,1000)IPASS
1000    FORMAT('  REQUEST COMPLETE!',2X,I6)
        IPASS=IPASS+1
        GO TO 5

C
C REPORT ANY ERRORS THAT HAVE BEEN UNCOVERED IN THE EXAMPLE.
C
100     WRITE(5,1010)ISTAT
1010    FORMAT(//,' ERROR ASSIGNING LUN TO LPA11-K   ',I6)
        CALL EXIT
110     WRITE(5,1020)ISTAT
1020    FORMAT(//,' ERROR STOPPING LPA11-K CLOCKA    ',I6)
        CALL EXIT
120     WRITE(5,1030)ISTAT
1030    FORMAT(//,' ERROR FROM SETIBF - OUTPUT BUFFER   ',I6)
        CALL EXIT
130     WRITE(5,1040)ISTAT
```

```
1040    FORMAT(//,' ERROR FROM RLSBUF - OUTPUT BUFFER   ',I6)
        CALL EXIT
140     WRITE(5,1050)ISTAT
1050    FORMAT(//,' ERROR FROM SETIBF - INPUT BUFFER   ',I6)
        CALL EXIT
150     WRITE(5,1060)ISTAT
1060    FORMAT(//,' ERROR FROM RLSBUF - INPUT BUFFER   ',I6)
        CALL EXIT
160     WRITE(5,1070)IBUFNO,(OUTIOS(I),I=1,4)
1070    FORMAT(//,' ERROR FROM DOSWP   ',I2,4(3X,O4))
C
C *** WARNING *** DISWP MIGHT STILL BE ACTIVE WHEN YOU EXIT
C
        CALL EXIT
170     WRITE(5,1080)IBUFNO,(INIOS(I),I=1,4)
1080    FORMAT(//,' ERROR FROM DISWP   ',I2,4(3X,O4))
C
C *** WARNING *** DOSWP MIGHT STILL BE ACTIVE WHEN YOU EXIT
C
        CALL EXIT
180     WRITE(5,1090)I,J,OUTBUF(I,J),INBUF(I,J)
1090    FORMAT(//,' *DATA ERROR* - WORD # ',I4,2X,I4,4X,O6,2X,O6)
        CALL EXIT
        END
```

Chapter 13

K-Series Peripheral Support Routines

13.1 Introduction to K-Series Peripheral Support Routines

K-series laboratory peripheral modules are supported through a set of program-callable routines that are linked with your task at task-build time. These routines are highly modular. Therefore, a particular task contains only that code necessary for the facilities that it actually uses. Additionally, the support routines perform I/O operations through the Connect to Interrupt Vector (CINT$) Executive directive. This directive allows your task to bypass normal QIO processing and perform I/O nearly independent of the Executive.

The following subsections briefly describe the K-series laboratory peripherals, the features provided by the K-series support routines, and the generation and use of these routines.

13.1.1 K-Series Laboratory Peripherals

The K-series peripheral support routines provide single-user, task-level support for the following laboratory peripheral modules:

- AA11-K digital-to-analog (D/A) converter
- AD11-K analog-to-digital (A/D) converter
- AM11-K multiple gain multiplexer
- DR11-K digital I/O interface
- KW11-K dual programmable real-time clock
- AAV11-A D/A converter (LSI-11-bus compatible)
- ADV11-A A/D converter (LSI-11-bus compatible)
- DRV11 parallel line unit (LSI-11-bus compatible)
- KWV11-A programmable real-time clock (LSI-11-bus compatible)

The maximum supported hardware configuration consists of one KW11-K and 16 of each of the AA11-K, AD11-K (with optional AM11-K), and DR11-K modules. The minimum configuration, if synchronous sweeps are desired, would be one KW11-K and any one of the three other modules. A single DR11-K supports nonclocked, interrupt-driven I/O sweeps or single digital input or output. A single AD11-K supports single-word A/D input and nonclocked, overflow-driven sampling (provided that the A/D conversion is started with the EXT start input on the AD11-K). An AA11-K supports burst mode output and scope control.

13.1.1.1 AA11-K D/A Converter

The AA11-K includes four 12-bit digital-to-analog (D/A) converters and an associated display control. The display control permits your task to display data in the form of a 4096 x 4096 dot array. Under program control, a dot may be produced at any point in this array, and a series of these dots may be programmed sequentially to produce graphic output. The display control may output to chart or X/Y recorder or cathode-ray tube (CRT) display unit.

The AAV11-A is an LSI-11-bus-compatible D/A converter with characteristics similar to those of the AA11-K.

13.1.1.2 AD11-K A/D Converter

The AD11-K is a 12-bit successive approximation converter that enables your task to sample analog data at specified rates and to store the equivalent digital value for subsequent processing. The basic subsystem consists of an input multiplexer (switch-selectable between 16-channel single-ended or 8-channel differential), sample-and-hold circuitry, and a 12-bit A/D converter. By changing jumpers, the analog inputs can be made bipolar or unipolar.

The ADV11-A is an LSI-11-bus-compatible D/A converter with characteristics similar to those of the AD11-K.

13.1.1.3 AM11-K Multiple Gain Multiplexer

The AM11-K is a multiplexer expander that supplements the 16-channel, single-end (8-channel differential) analog input multiplexer in the AD11-K. The expansion is done in three independent groups on the AM11-K. Each group can be set to 16 single-ended or pseudo-differential or 8 differential input channels; each group can have a gain of 1, 4, 16, or 64 assigned to it by a switch in the amplifier.

13.1.1.4 DR11-K Digital I/O Interface

The DR11-K is a general-purpose digital I/O interface capable of parallel transfer of up to 16 bits of data, under program control, between a PDP–11 UNIBUS computer and an external device (or another DR11-K).

The DRV11 is an LSI-11-bus-compatible, general-purpose I/O interface with characteristics similar to those of the DR11-K.

13.1.1.5 KW11-K Dual Programmable Real-Time Clock

The KW11-K is a dual programmable real-time clock option that PDP–11 UNIBUS computers use. The KWV11-A is an LSI-11-bus-compatible real-time clock with characteristics similar to those of the KW11-K.

Clock A

Clock A features are as follows:

- A 16-bit counter
- A 16-bit programmable preset/buffer register
- Four modes of operation
- Two external inputs (Schmitt Triggers)
- Eight clock rates, program selectable
- Five clock frequencies, crystal controlled for accuracy
- Processor actions synchronized to external events

Clock B

Clock B features are as follows:

- An 8-bit counter
- An 8-bit programmable preset register
- Repeated interval mode of operation
- One external input (Schmitt Trigger)
- Seven clock rates, program selectable
- Five clock frequencies, crystal controlled for accuracy

13.1.2 Support Routine Features

The RSX–11M–PLUS program-callable K-series support routines provide the following features:

- Clock overflow or trigger-driven A/D sweep
- Clock overflow or interrupt-driven digital input sweep
- Clock overflow or interrupt-driven digital output sweep
- Clock overflow or burst mode D/A sweep
- Single digital input
- Single digital output
- Single A/D input
- Scope control
- Histogram sampling

- Schmitt Trigger simulation
- Clock control
- A 16-bit software clock
- A/D input to real number conversion
- Buffer control

Immediate digital input or output can be performed at any time. Multiple clock-driven sweeps can be initiated if this optional feature was selected during the K-series generation dialog (see Section 13.1.3.1). Such sweeps, however, are subject to the following restrictions:

- Regardless of the number of controllers present, there can be only one active A/D sweep at any point in time. The same restriction holds true for D/A sweeps. It is possible, however, to perform digital input and digital output sweeps simultaneously, using the same DR11-K, so long as this feature is selected during the generation dialog.
- There can be no conflict in clock rates among the sweeps.
- Only the first sweep can use the delay from start event.
- The interevent time data-gathering routine cannot run in parallel with any other clock-driven sweeps.

13.1.3 Generation and Use of K-Series Routines

To use K-series support routines, you must do the following three things during system generation:

- Reserve necessary vector space.
- Specify that the CINT$ Executive directive is to be included in the system.
- Specify that asynchronous system trap (AST) support is required.

After system generation, you must follow particular procedures for the generation of K-series support routines and the program use of K-series routines

These two procedures are detailed in the following subsections.

13.1.3.1 Generation of K-Series Support Routines

An indirect command file, similar to those that system generation uses, generates the K-series support routine library and other necessary facilities. You invoke this command file by entering the following command line.

```
>@[200,200]SGNKLAB RET
```

The dialog initiated by this command determines the following:

- The device configuration of the subsystem
- The maximum number of buffers that are used on a per-sweep basis
- The inclusion or omission of optional features such as multiple clock-driven sweeps and duplex digital I/O sweeps

After this information is obtained, the command file creates the following:

1. A prefix file, [45,10]KPRE.MAC, for use during assembly of K-series support routines.
2. A database file, [45,10]KIODT.MAC, that contains control blocks needed to support the devices.
3. A common block file, [45,10]KCOM.MAC, that allows your tasks to access the I/O page. Use this file only on mapped systems.
4. On mapped systems only, two indirect command files:
   a. A file, [45,24]KCOMBLD.CMD, which is a Task Builder (TKB) build file for the common block
   b. A file, [1,54]INSKCOM.CMD, that installs the common block

At your option, the K-series routines themselves can then be assembled and an object library can be created. You can specify the name of this library or accept the following default file specification:

```
LB:[1,1]KLABLIB.OLB
```

13.1.3.2 Program Use of K-Series Routines

The steps required for routine program use of K-series support routines are as follows:

1. Compile or assemble the program. If the task is overlaid, you must ensure that both the buffers used by the K-series support routines and that the support routines themselves reside in the root section of the overlay structure.
2. Invoke TKB as follows:
   a. On mapped systems only, use the /PR:0 switch to indicate that the task is privileged.
   b. Include the following indirect command among the responses to the TKB prompt where x is 0 for unmapped systems and 4 for mapped systems:

      ```
      TKB>@[1,5x]LNK2KLAB
      ```

   c. On mapped systems only, enter the following indirect command in response to the prompt for options:

      ```
      ENTER OPTIONS
      @[1,54]LNK2KCOM RET
          .
          .
          .
      //
      ```

3. On mapped systems only, enter the following indirect command from a privileged terminal before executing the program:

   ```
   >@[1,54]INSKCOM RET
   ```

Example

```
>F4P KTEST,KTEST/-SP=KTEST [RET]
>TKB [RET]
TKB>KTEST/PR:0,KTEST/-SP=KTEST,[1,1]F4POTS/LB [RET]
TKB>@[1,54]LNK2KLAB [RET]
TKB>/ [RET]
ENTER OPTIONS
TKB>@[1,54]LNK2KCOM [RET]
    .
    .
    .
TKB>// [RET]
```

13.2 The Program Interface

A collection of program-callable subroutines provides access to the K-series laboratory peripherals. The formats of these calls are fully documented here for FORTRAN programs. MACRO-11 programmers access these same subroutines either through the standard subroutine linkage or through the use of two special-purpose macros. Both techniques are described in Section 13.2.2. Both FORTRAN and MACRO programs must contain at least one I/O status block (IOSB), described in Section 13.2.3, for retrieval of status information.

13.2.1 FORTRAN Interface

Table 13-1 lists the FORTRAN interface subroutines for accessing K-series laboratory peripherals.

Table 13-1: FORTRAN Subroutines for K-Series Laboratory Peripherals

Subroutine	Function
ADINP	Initiate single analog input
ADSWP	Initiate synchronous A/D sweep
CLOCKA	Set Clock A rate
CLOCKB	Control Clock B
CVADF	Convert A/D input to floating point
DASWP	Initiate synchronous D/A sweep
DIGO	Digital start event
DINP	Digital input
DISWP	Initiate synchronous digital input sweep
DOSWP	Initiate synchronous digital output sweep
DOUT	Digital output
FLT16	Convert unsigned integer to a real constant
GTHIST	Gather interevent time data

Table 13–1 (Cont.): FORTRAN Subroutines for K-Series Laboratory Peripherals

Subroutine	Function
IBFSTS	Get buffer status
ICLOKB	Read 16-bit clock
IGTBUF	Return buffer number
INXTBF	Set next buffer
IWTBUF	Wait for buffer
RCLOKB	Read 16-bit clock
RLSBUF	Release data buffer
RMVBUF	Remove buffer from device queue
SCOPE	Control scope
SETADC	Set channel information
SETIBF	Set array for buffered sweep
STPSWP	Stop sweep
XRATE	Compute clock rate and preset

The calling sequences of the routines listed in Table 13–1 are compatible with the routines for the LPA-11, described in Chapter 12. The following subsections briefly describe the function and format of each FORTRAN subroutine call.

13.2.1.1 ADINP—Initiate Single Analog Input

The ADINP routine obtains a single word as input from the A/D converter.

ADINP can be invoked as a subroutine or a function.

Formats

```
CALL ADINP   ([iflag],[ichan],ival)

ival=IADINP  ([iflag],[ichan],[ival])
```

Parameters

iflag

Specifies the following gain options:

0 Absolute channel addressing (default). This is the only mode supported on the ADV11 (Q-bus).

1 Sample at a gain of 1. In modes 1, 2, 3, 4, and 5 each AD11-K/AM11-K is treated as 16 channels with channels 17 to 63 strapped to gains 4, 16, and 64. The 48 multiplexer channels are selected by the software according to the gain specification. Mode values 1, 2, 3, 4, and 5 are not supported on the ADV11 (Q-bus version).

2 Sample at a gain of 4.

3 Sample at a gain of 16.

4 Sample at a gain of 64.

5 Perform auto-gain ranging.

ichan

Selects the channel to be sampled. The default is 0.

ival

Receives the sample. The gain bits are inserted if iflag is nonzero.

13.2.1.2 ADSWP—Initiate Synchronous A/D Sweep

The ADSWP routine initiates a synchronous A/D input sweep through an AD11-K (and, if present, the AM11-K). The analog input word placed in your task's buffer consists of the 12 bits read from the A/D converter and (except when the mode parameter equals 0) the 2 gain bits read from the A/D status register. A value of 177776_8 is returned for A/D timeout. A value of 177777_8 is returned on an A/D conversion error. Such errors are typically caused by conversions occurring too fast.

If differential input is desired, the channel increment must be set to 2 by calling the SETADC routine. The default channel increment is 1 (single-ended input).

Note

This routine expects to have the ST1 OUT from the KW11-K or similar trigger jumped to EXT START on the AD11-K if mode 512 is desired. This also requires the A EVENT OUT from the KW11-K clock trigger jumped to the KW overflow on the AD11-K if clock driven sweeps are desired.

Format

CALL ADSWP (ibuf,lbuf,[nbuf],[mode],[iprset],[iefn],[ldelay],[ichn],[nchn])

Parameters

ibuf

Specifies a 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

Specifies the size, in words, of each data buffer. All data buffers must be equal in size and lbuf must be greater than 0.

nbuf

Specifies the number of buffers to be filled. If nbuf is omitted or set equal to 0, indefinite sampling occurs. The STPSWP routine terminates indefinite sampling.

mode

Specifies sampling options. The default is 0. The mode bit values in the following list that are preceded by a plus sign (+) are independent and can be ADDed or ORed together. Those values not preceded by a plus sign are mutually exclusive and you can use only one such value at a time. All bit values that are not in the following list are reserved.

The following values can be specified:

0 Absolute channel addressing (default). This mode allows your task to directly access all 63 channels of an AD11-K/AM11-K combination. This is the only mode that is LPA-11 compatible.

1 Sample with a gain of 1. In modes 1, 2, 3, 4, and 5 each AD11-K/AM11-K is treated as 16 channels with channels 17 to 63 strapped to gains 4, 16, and 64. The 48 multiplexer channels are selected by the software according to the gain specification. Mode values 1, 2, 3, 4, and 5 are not supported on the ADV11 (Q-bus version).

2 Gain of 4. See also mode value 1.

3 Gain of 16. See also mode value 1.

4 Gain of 64. See also mode value 1.

5 The driver executes auto-gain ranging to return the result with the most significance. Note that using auto-gain ranging may require dual sampling and this impacts performance. See also mode value 1.

7 The driver uses an interrupt routine that you supply. The routine must be named .ADINU and must follow the interrupt service routine coding conventions that this subsystem uses. Refer to the source module KADIN5.MAC for an example of an A/D interrupt routine.

+256 External start (ST1).

+512 Nonclock overflow sampling triggered by ST1.

iprset

Specifies the clock preset. The clock rate divided by the clock preset value yields the clock overflow rate. You can use the XRATE subroutine to calculate a clock preset value. If you omit the iprset argument from the ADSWP call, you must specify a mode value of +512. Otherwise, the driver returns an error status code of 301 (invalid arguments) into the IOSB.

iefn

Specifies the event flag (1 to 96), a completion routine, or 0. If iefn is 0 or is defaulted, event flag 30 is used for internal synchronization. If iefn is an event flag (1 to 96), the selected event flag is set as each buffer is filled. If iefn is greater than 96, it is considered

to be a completion routine that is called with a jump to subroutine program counter (JSR PC). Such routines must return with an RTS PC (or a FORTRAN RETURN statement). Furthermore, FORTRAN completion routines must not do any I/O through the FORTRAN run-time system because this may cause unpredictable results or fatal task errors.

If multiple sweeps are initiated, you should specify different event flags. Adherence to this limitation cannot be enforced by the software.

Idelay

Specifies the delay from the start event (ST1) until the first sample in IRATE units. The default or 0 indicates no delay.

ichn

Specifies the number of the first channel to be sampled. The default of 0 applies only if ichn was not established in a prior call to the SETADC routine.

nchn

Specifies the number of channels to sample. The default is 1. The parameter nchn may be set up with the SETADC routine. All nchn channels are sampled on one clock interrupt.

13.2.1.3 CLOCKA—Set Clock A Rate

The CLOCKA routine sets the rate for Clock A.

Format

CALL CLOCKA (irate,iprset,[ind],[lun])

Parameters

irate

Specifies the clock rate. One of the following must be specified:

0 Clock B overflow (not on Q-bus version) or no rate

1 1 megahertz (MHz)

2 100 kilohertz (kHz)

3 10 kHz

4 1 kHz

5 100 hertz (Hz)

6 Schmitt Trigger 1

7 Line frequency

iprset

Specifies the clock preset. The clock rate divided by the clock preset value yields the clock overflow rate. You can use the XRATE routine to calculate a clock preset value. The two's complement of this value is the one that you must supply.

ind

Receives a success or failure code as follows:

0 Indicates illegal arguments.

1 Indicates Clock A is set to start when sweep requested.

lun

Specifies the logical unit number. The argument is ignored by the K-Series routines. However, it is present for LPA-11 compatibility.

13.2.1.4 CLOCKB—Control Clock B

The CLOCKB routine gives you control over the KW11-K Clock B, which maintains a 16-bit software clock. This feature is not available on LSI-11-bus versions. The 16-bit clock is incremented once per Clock B interrupt. The maximum value of the clock is 65535.

Format

CALL CLOCKB ([irate],[iprset],[mode],[ind],[lun])

Parameters

irate

Specifies the clock rate. When irate is nonzero, the clock is set running at the selected rate after the preset value specified by iprset is loaded. The 16-bit software clock is not altered by starting the clock. The initial value of the 16-bit clock is 0 when the program is loaded.

When irate is 0, clock B is stopped but the 16-bit software clock is unaltered.

When irate is defaulted, the 16-bit software clock is zeroed but clock B continues to run.

The following are the acceptable values for irate:

0 Stop Clock B

1 1 MHz

2 100 kHz

3 10 kHz

4 1 kHz

5 100 Hz

6 Schmitt Trigger 3

7 Line frequency

iprset

Specifies the count by which to divide clock rate to yield overflow rate. You can use overflow events to maintain the 16-bit software clock or to drive clock A, or both. The default value is 1. The maximum practical overflow rate in interrupt mode is 10 kHz. The range of iprset is 1 to 255. The value in iprset can be established by use of the XRATE routine.

mode

Specifies the options. Either of the following can be specified:

0 Indicates normal operations. This is the default. The 16-bit software clock is updated on Clock B overflow. The overflow rate should not exceed 10 kHz. The software does not check the overflow rate.

1 Indicates Clock B operates in noninterrupt mode. The 16-bit clock is not incremented or altered. This allows a greater than 10-kHz pulse to be sent to Clock A.

ind

Receives a success or failure code as follows:

0 Indicates a failure to start Clock B.

1 Indicates Clock B started.

lun

Specifies the logical unit number. This argument is ignored by the K-series routines. It is present for LPA-11 compatibility.

13.2.1.5 CVADF—Convert A/D Input to Floating Point

The CVADF routine converts an A/D input value to a floating-point number. The routine can be invoked as a subroutine or a function.

Formats

```
CALL CVADF   (ival,val)

val = CVADF(ival)
```

Parameters

ival

Specifies a value obtained from A/D input. Bits 12 to 15 are the gain. Bits 0 to 11 represent the value.

val

Receives the converted value (REAL*4).

13.2.1.6 DASWP—Initiate Synchronous D/A Sweep

The DASWP routine initiates synchronous D/A output to an AA11-K.

Format

```
CALL   DASWP (ibuf,lbuf,[nbuf],[mode],[iprset],[iefn],[ldelay],[ichn],[nchn])
```

Parameters

ibuf

Specifies a 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

Specifies the size, in words, of each data buffer. All data buffers must be equal in size and lbuf must be greater than 0.

nbuf

Specifies the number of buffers to be emptied. If nbuf is omitted or set equal to 0, indefinite emptying occurs. The STPSWP routine terminates indefinite emptying.

mode

Specifies the start criteria. Except where noted, the plus signs (+) preceding mode bit values in the following list indicate that the values are independent and can be ADDed or ORed together. All bit values that are not in the following list are reserved.

The following values can be specified:

0 Indicates immediate start. This is the default.

1 Indicates that a group of data words, whose number is specified by nchn, is preceded by a scope control word (refer to Section 13.2.1.22 for a description of scope control words). This bit setting is ignored if +512 is also specified. This feature is not included in the Q-bus (AAV11) version.

The buffer size specified by lbuf must be a multiple of nchn+1 words. The DASWP routine, however, does not enforce this restriction.

2 Sets the intensify bit after each pair of channels (nchn must be 2) have been output. This feature is supported on the Q-bus version only. It assumes that bit 0 of DAC3 on the AAV11 is connected to the intensify input on the oscilloscope.

+256 Indicates external start (ST1).

+512 Indicates non-clock-overflow, non-interrupt-driven output (burst mode). This value cannot be specified with either external start (+256) or a nonzero ldelay value. A completion routine must be specified if nbuf is greater than the number of buffers supplied or if continuous burst output is desired. If nbuf equals -1, burst mode must be stopped by calling STPSWP from the completion routine.

iprset

Specifies the clock preset. The clock rate divided by the clock preset value yields the clock overflow rate. You can use the XRATE subroutine to calculate a clock preset value.

If the iprset parameter is omitted, you must specify a mode value of +512. Otherwise, an error status code of 301 (invalid arguments) is returned into the IOSB.

iefn

Specifies an event flag number (from 1 to 96), a completion routine, or 0. If you use 0 or default this value, the driver uses event flag 30 for internal synchronization. If iefn is an event flag from 1 to 96, the driver sets the selected event flag as each buffer is emptied. If iefn is greater than 96, the driver considers it a completion routine and calls it with a

JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement). Furthermore, FORTRAN completion routines must not perform I/O through the FORTRAN run-time system because this may cause unpredictable results or fatal task errors.

If multiple sweeps are initiated, you should specify different event flags. This limitation cannot be enforced by the software.

idelay

Specifies the delay from start event (ST1) until the first sample in irate units. The default or 0 indicates no delay.

ichn

Specifies the first channel number. The default is 0.

nchn

Specifies the number of channels. The default is 1. When nchn equals 2 and mode does not contain +1, the size of data buffers specified in lbuf must be an even number. The software does not check this requirement.

13.2.1.7 DIGO—Digital Start Event

The DIGO routine allows you to specify the digital input bits that, when set, cause the simulation of an external start event and the start of a pending sweep.

Format

CALL DIGO ([iunit],[mask],[kount])

Parameters

iunit

Specifies the DR11-K unit number. The default is 0.

mask

Specifies a logical mask that specifies one or more start bits. If zero, a pending digital start event request is immediately canceled. If mask is defaulted, an ST1 event is immediately simulated and the current value of the 16-bit software clock is returned, if kount is specified.

kount

Receives the current value of the 16-bit software clock when the defaulting of mask causes the simulation of an ST1 event.

13.2.1.8 DINP—Digital Input

The DINP routine inputs a single 16-bit word from a DR11-K. Bits read as a 1, can be masked with a 1, which causes the clearing of the bit in the DR11-K input buffer.

During the K-series routines generation dialog, it is possible to select one of the following two versions of the DINP routine:

- A slow version containing all functions described in the following sections
- A fast version that omits the functions provided by the mask, iosb, and input arguments

The fast version of DINP can be invoked as a function (IDINP) only. The slow version of DINP can be invoked as a subroutine or a function.

Formats

CALL DINP ([iunit],[mask],iosb,input)

ind = IDINP(iunit,[mask],iosb,[input])

Parameters

iunit

Specifies the DR11-K unit number. This argument is required for the fast version of DINP. For the slow version, the default is 0.

mask

Specifies the bit mask that specifies which input bits are cleared in the digital input register. The default is 177777_8, which indicates all bits are cleared.

iosb

Specifies a 2-word I/O status block array (see Section 13.2.3).

input

Receives the data input from the DR11-K.

ind

Receives the data input from the DR11-K if DINP is invoked as a function.

13.2.1.9 DISWP—Initiate Synchronous Digital Input Sweep

The DISWP routine initiates a synchronous digital input sweep through a DR11-K.

Format

CALL DISWP (ibuf,lbuf,[nbuf],[mode],[iprset],[iefn],[ldelay],[iunit])

Parameters

ibuf

Specifies a 40-word array initialized by the SETIBF routine. The first two words of the array are the IOSB.

lbuf

Specifies the size, in words, of each data buffer. All data buffers must be equal in size and lbuf must be greater than 0.

nbuf

Specifies the number of buffers to be filled. If nbuf is 0 or is defaulted, indefinite sampling occurs. The STPSWP routine terminates indefinite sampling.

mode

Specifies the sampling options. The default is 0. The plus signs (+) preceding the mode bit values in the following list indicate that the values are independent and can be added or ORed together.

The following values can be specified:

0	Single-word sample, immediate start. This is the default mode.
+256	External start (ST1).
+512	Nonclock overflow interrupt-driven input. External start and delay are illegal.
+1024	Time-stamped sampling. The doubleword consists of one data word followed by the value of the 16-bit software clock at the time of the sample. This option is not available if you are not using the KW11-K clock (for example, on the Q-bus).

iprset

Specifies the clock preset. The clock rate divided by the clock preset value yields the clock overflow rate. You can use the XRATE subroutine to calculate a clock preset value.

If the iprset argument is omitted, you must specify a mode value of +512. Otherwise, an error status code of 301 (invalid arguments) is returned to the IOSB.

iefn

Specifies an event flag number (from 1 to 96), or a completion routine, or 0. If you use 0 or default this value, the driver uses event flag 30 for internal synchronization. If you use iefn as an event flag from 1 to 96, the driver sets the selected event flag as each buffer is filled. If iefn is greater than 96, the driver considers the iefn a completion routine and calls it with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement). Furthermore, FORTRAN completion routines must not perform I/O through the FORTRAN run-time system because this may cause unpredictable results or fatal task errors.

If multiple sweeps are initiated, you should specify different event flags. This limitation cannot be enforced by the software.

ldelay

Specifies the delay from start event (ST1) until the first sample in irate units. The default or 0 indicates no delay.

iunit

Specifies the DR11-K unit number. The default is 0.

13.2.1.10 DOSWP—Initiate Synchronous Digital Output Sweep

The DOSWP routine initiates a synchronous digital output sweep through a DR11-K.

Format

```
CALL  DOSWP (ibuf,lbuf,[nbuf],[mode],[iprset],[iefn],[ldelay],[iunit])
```

Parameters

ibuf

Specifies a 40-word array initialized by the SETIBF routine. The first two words of the array are the IOSB.

lbuf

Specifies the size, in words, of each data buffer. All data buffers must be equal in size, and lbuf must be greater than 0.

nbuf

Specifies the number of buffers to be emptied. If nbuf is 0 or is defaulted, indefinite emptying occurs. The STPSWP routine terminates indefinite emptying.

mode

Specifies the start criteria. The default is 0.

The following values can be specified in the high-order byte of mode:

0 Immediate start. This is the default.

+256 External event start (ST1).

+512 Nonclock overflow, interrupt-driven output. External start and delay are illegal.

iprset

Specifies the clock preset. The clock rate divided by the clock preset value yields the clock overflow rate. You can use the XRATE subroutine to calculate a clock preset value.

If the iprset argument is omitted, you must specify a mode value of +512. Otherwise, an error status code of 301 (invalid arguments) is returned into the IOSB.

iefn

Specifies an event flag number (from 1 to 96), a completion routine, or 0. If you use 0 or default this value, the driver uses event flag 30 for internal synchronization. If iefn is an event flag from 1 to 96, the driver sets the selected event flag as each buffer is emptied. If iefn is greater than 96, the driver considers it a completion routine and calls it with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN statement). Furthermore, FORTRAN completion routines must not perform I/O through the FORTRAN run-time system because this may cause unpredictable results or fatal task errors.

If multiple sweeps are initiated, you should specify different event flags. This limitation cannot be enforced by the software.

idelay

Specifies the delay from start event (ST1) until the first sample in irate units. The default or 0 indicates no delay.

iunit

Specifies the DR11-K unit number. The default is 0.

13.2.1.11 DOUT—Digital Output

The DOUT routine outputs a single 16-bit word to a DR11-K. Only those bits in the output word specified by corresponding bits in a mask field are altered.

During the K-series routines generation dialog, it is possible to select one of the following two versions of the DOUT routine:

- A slow version containing all functions described in the following sections
- A fast version that omits the functions provided by the mask and iosb arguments

The slow version of DOUT can be invoked as a subroutine or a function. The fast version of DOUT can be invoked as a subroutine only.

Formats

```
CALL  DOUT ([iunit],[mask],iosb,idata)

iout = IDOUT([iunit],[mask],iosb,idata)
```

Parameters

iunit
Specifies the DR11-K unit number. The default is 0.

mask
Selects which bits can be altered. The default is 177777_8, indicating all bits.

iosb
Specifies a 2-word I/O status block (see Section 13.2.3).

idata
Specifies the 16-bit output value for the DR11-K. A 1 sets a corresponding bit. A 0 clears the corresponding bit.

iout
Receives a copy of the DR11-K output register after it has been altered.

13.2.1.12 FLT16—Convert Unsigned Integer to a Real Constant

The FLT16 routine converts an unsigned 16-bit integer to a real constant (REAL*4). It can be invoked as a subroutine or a function.

Formats

```
CALL  FLT16 (ival,val)

val = FLT16(ival[,val])
```

Parameters

ival

Specifies an unsigned 16-bit integer.

val

Specifies the converted (REAL*4) value.

13.2.1.13 GTHIST—Gather Interevent Time Data

The GTHIST routine initiates sampling to measure the elapsed time between events. The value of the Clock A buffer/preset register at the time of ST2 firing is stored in a task buffer that you provide.

GTHIST is an optional facility that must be explicitly selected during the K-series generation dialog prior to its use in any program.

Format

CALL GTHIST (ibuf,lbuf,[nbuf],[mode],[iprset],[iefn],[kount])

Parameters

ibuf

Specifies a 40-word array initialized by the SETIBF routine. The first two words of the array are the I/O status block (IOSB).

lbuf

Specifies the size, in words, of each data buffer. All data buffers must be equal in size and lbuf must be greater than 0.

nbuf

Specifies the number of buffers to be filled. If nbuf is 0 or is defaulted, indefinite sampling occurs. The STPSWP routine terminates indefinite sampling.

mode

Specifies the following sampling options:

0 Indicates external event timing without zero base. This is the default.

1 Indicates external event timing with zero base. This is the only mode supported for the KWV11.

iprset

Specifies a null argument. This parameter is present only to maintain compatibility with other sweep routine calling sequences.

iefn

Specifies an event flag number (from 1 to 96), a completion routine, or 0. If you use 0 or default this value, the driver uses event flag 30 for internal synchronization. If iefn is an event flag from 1 to 96, the driver sets the selected event flag as each buffer is filled. If iefn is greater than 96, the driver considers it a completion routine and calls it with a JSR PC. Such routines must return with an RTS PC instruction (or a FORTRAN RETURN

statement). Furthermore, FORTRAN completion routines must not perform I/O through the FORTRAN run-time system because this may cause unpredictable results or fatal task errors.

If multiple sweeps are initiated, you should specify different event flags. This limitation cannot be enforced by the software.

kount

Specifies a counter used by GTHIST, as described in the following paragraphs.

To take Post-Stimulus Time data, set mode to 0. ST1 signals the occurrence of a stimulus and starts the clock (that is, no data is taken until the first ST1 occurs). Each response is signaled by ST2, and the buffer/preset register contents are placed in your task's buffer. Each ST1 resets the counter register to 0, and increments kount by 1. Thus, kount keeps track of the number of stimuli (ST1 events). Clock overflow stops the clock. The clock waits for the next ST1 event before restarting. The maximum stimulus-response interval is a function of the clock rate.

To obtain Inter-Stimulus-Interval data, set mode to 1. The time between successive events, as signaled by ST2, is recorded. The maximum interevent time is a function of the clock rate. When clock overflow occurs, the value returned on the next ST2 firing is 177777_8 and kount is incremented. Thus, kount represents the number of times the maximum interevent time was exceeded. In general, the user should ignore values of 177777_8.

13.2.1.14 IBFSTS—Get Buffer Status

The IBFSTS routine returns information on buffers that the driver is using in a sweep.

Format

```
CALL  IBFSTS (ibuf,istat)
```

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

istat

Specifies an array with as many elements as there are buffers involved in the sweep. The maximum is eight. IBFSTS fills each element in the array with the status of the corresponding buffer. The possible status codes are as follows:

+2 Indicates that the buffer is in the device queue. That is, it is waiting to be filled or emptied.

+1 Indicates that the buffer is in the task queue. That is, it is full of data (for input sweeps) or is waiting to be filled (for output sweeps).

0 Indicates that the status of the buffer is unknown. That is, it is not the current buffer nor is it in either the device or the user task queue.

-1 Indicates that a service routine is currently using the buffer.

13.2.1.15 ICLOKB—Read 16-Bit Clock

The ICLOKB function returns the contents of the 16-bit software clock as an integer value to your task.

Format

```
CALL   itim = ICLOKB(0)
```

Parameter

itim

Receives the current value of the 16-bit software clock as an unsigned integer.

Note

MACRO–11 programmers need not establish an argument block for the ICLOKB function. The current value of the software clock is returned in R0.

13.2.1.16 IGTBUF—Return Buffer Number

The IGTBUF routine returns the number of the next buffer to use. This routine should be called by your task's completion routines to determine the next buffer to access. Do not use it if an event flag was specified in the sweep-initiating call. Rather, use the IWTBUF routine with event flags.

IGTBUF can be invoked as a subroutine or a function.

Formats

```
CALL   IGTBUF (ibuf,ibufno)


ibufno = IGTBUF(ibuf[,ibufno])
```

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

ibufno

Receives the number of the next buffer to access. If there is no buffer in the queue, ibufno contains -1.

13.2.1.17 INXTBF—Set Next Buffer

The INXTBF routine alters the normal buffer selection algorithm. It allows your task to specify the number of the next buffer to be filled or emptied.

INXTBF can be invoked as a subroutine or a function.

Formats

CALL INXTBF (ibuf,ibufno[,ind])

ind = INXTBF(ibuf,ibufno[,ind])

Parameters

ibuf

Specifies the 40-word array specified in the call that initiated a sweep.

ibufno

Specifies the number of the next buffer that the task wants filled or emptied. The buffer must already be in the device queue.

ind

Receives an indication of the result of the operation as follows:

0 Indicates that the specified buffer was already active or was not in the device queue.

1 Indicates that the next buffer was set successfully.

13.2.1.18 IWTBUF—Wait for Buffer

The IWTBUF routine allows your task to wait for the next buffer to fill or empty. Use it with an event flag specified in in the sweep-initiating call. Do not use this routine if you specified a completion routine in the call to initiate a sweep. Rather, use the IGTBUF routine with completion routines.

IWTBUF can be invoked as a subroutine or a function.

Formats

CALL IWTBUF (ibuf,[iefn],ibufno)

ibufno = IWTBUF(ibuf,[iefn],[ibufno])

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

iefn

Specifies the event flag for which the task waits. This should be the same event flag as that specified in the sweep-initiating call. If iefn equals 0 or is defaulted, the driver uses event flag 30.

ibufno

Receives the number of the next buffer to be filled or emptied by your task.

13.2.1.19 RCLOKB—Read 16-Bit Clock

The RCLOKB routine returns to your task the contents of the 16-bit software clock as a real constant.

RCLOKB can be invoked as a subroutine or a function.

Formats

```
CALL  RCLOKB (rlast,time)

time=RCLOKB(rlast,time)
```

Parameters

time

Receives the current value of the 16-bit software clock as a real constant (REAL*4).

rlast

Specifies a value (REAL*4) to be subtracted from the current 16-bit software clock before it is returned into the time field.

13.2.1.20 RLSBUF—Release Data Buffer

The RLSBUF routine declares one or more buffers free for use by the interrupt service routine.

The RLSBUF routine must be called to the release buffer or buffers to the device queue before the sweep is initiated. The device queue must always contain at least one buffer to maintain continuous sampling. Otherwise, buffer overrun occurs (see Section 13.3 for a discussion of buffer management). Note that RLSBUF does not verify whether the specified buffers are already in a queue.

Format

```
CALL  RLSBUF (ibuf,ind,n0[,n1...,n7])
```

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

ind

Receives a success or failure code as follows:

0 Indicates illegal buffer number specified.

1 Indicates buffer or buffers successfully released.

n0,n1,...,n7

Specifies the numbers of buffers to be released. A maximum of eight can be specified.

13.2.1.21 RMVBUF—Remove Buffer from Device Queue

The RMVBUF routine removes a buffer from the device queue.

Format

CALL RMVBUF (ibuf,n[,ind])

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

n

Specifies the number of the buffer to remove.

ind

Receives a success or failure code as follows:

0 Indicates that the specified buffer was not in the device queue.

1 Indicates that the specified buffer was removed from the queue.

13.2.1.22 SCOPE—Control Scope

The SCOPE routine allows you to control the status register of an AA11-K

Format

CALL SCOPE (iunit,icntrl,iosb)

Parameters

iunit

Specifies the AA11-K unit number.

icntrl

Specifies a combination of bit values as shown in Table 13–2. Any bits not listed in this table are cleared before output to the AA11-K status register

iosb

Specifies a 2-word I/O status block (see Section 13.2.3).

Table 13–2: Scope Control Word Values

Decimal Value	Octal Value	Function
4096	10000	Erase storage CRT
2048	4000	Set write-through mode
1024	2000	Set store mode
512	1000	A digital signal available in the AA11-K.

Table 13–2 (Cont.): Scope Control Word Values

Decimal Value	Octal Value	Function
12	14	Intensify on X or Y
8	10	Intensify on Y
4	4	Intensify on X
2	2	Fast intensify enable
1	1	Intensify pulse

The values in Table 13–2 also create scope control words for calls to the DASWP routine with a mode value of 1.

13.2.1.23 SETADC—Set Channel Information

The SETADC routine establishes channel start and increment information for an A/D sweep.

SETADC can be invoked as a subroutine or a function as follows:

Format

```
CALL  SETADC (ibuf,[iflag],[ichn],[nchn],[inc],[ind])
```

Format

```
ind=ISTADC (ibuf,[iflag],[ichn],[nchn],[inc],[ind])  1
```

Parameters

ibuf

Specifies a 40-word array initialized by the SETIBF routine.

iflag

Equals zero if you want absolute addressing and equals nonzero for programmable gain addressing. The default is 0.

ichn

Specifies the first channel number. The default is 0.

nchn

Specifies the number of samples to be taken per interrupt. The default is 1.

inc

Specifies the channel increment. The default is 1. You should specify an increment of 2 for differential A/D input.

ind

Receives a success or failure code as follows:

0 Indicates an illegal channel number.

1 Indicates successful recording of channel information for an A/D sweep.

13.2.1.24 SETIBF—Set Array for Buffered Sweep

The SETIBF routine initializes an array required by buffered sweep routines.

Format

CALL SETIBF (ibuf,[ind],[lamskb],buf0[,buf1...buf7])

Parameters

ibuf

Specifies a 40-word array.

ind

Receives a success or failure code as follows:

0 Indicates an illegal number of buffers was specified. SETIBF initializes the array according to the maximum number of buffers allowed. You specify this maximum number of buffers during the K-series system generation dialog.

1 Indicates the array was initialized successfully.

lamskb

Specifies a 4-word array. It is present for compatibility with LPA-11 routines, but it is ignored by K-series software.

buf0,...,buf7

Specifies the name of a buffer. A maximum of eight buffers can be specified. Any buffer names in excess of eight are ignored. At least two buffers must be specified to maintain continuous sampling.

Each buffer specified in the call to SETIBF is assigned a number from 0 to 7.

The assignment of these numbers is based on the order in which buffer names appear in the argument list. The first buffer whose name appears in the list is assigned number 0, the second is assigned number 1, and so forth. In all subsequent calls to other K-series routines involving the set of buffers specified in a call to SETIBF, these numbers, rather than names, refer to particular buffers.

13.2.1.25 STPSWP—Stop Sweep

The STPSWP routine allows your task to stop a sweep that is in progress.

Format

CALL STPSWP (ibuf[,iwhen],[ind])

Parameters

ibuf

Specifies the 40-word array in the call that initiated a sweep.

iwhen

Specifies when to stop the sweep as follows:

0 Indicates the next sample. This is the default.

+n Indicates the end of the current buffer. (Any positive value)

-n Reserved. (Any negative value)

ind

Receives a success or failure code as follows:

0 Indicates that the sweep was not active or no sweep could be found that was associated with the specified ibuf.

1 Indicates that the sweep is stopped (at the time indicated by iwhen).

13.2.1.26 XRATE—Compute Clock Rate and Preset

The XRATE routine computes an appropriate clock rate and preset that achieves a desired dwell (intersample interval).

Note

You can use the XRATE routine only on systems that have a FORTRAN or BASIC-PLUS-2 compiler.

XRATE can be invoked as a subroutine or a function as follows:

Format

```
CALL  XRATE (dwell,irate,iprset,iflag)
```

Format

```
adwell=XRATE (dwell,irate,iprset,iflag)  1
```

Parameters

dwell

Specifies the intersample time that you want. The time is expressed in decimal seconds (REAL*4).

irate

Receives the computed clock rate as a value from 1 to 5.

iprset

Receives the clock preset.

iflag

Specifies whether the computation is for Clock A or Clock B as follows:

0	Indicates the computation is for Clock A.
Nonzero	Indicates the computation is for Clock B.

adwell

Specifies the actual dwell rate for the clock based on the irate and iprset parameters.

13.2.2 MACRO–11 Interface

MACRO–11 programmers access the K-series support routines described in Section 13.2.1 through either of the following two techniques:

- The standard subroutine linkage mechanism and the CALL op code
- Special-purpose macros that generate an argument list and invoke a subroutine

These techniques are described in the following subsections.

13.2.2.1 Standard Subroutine Linkage and CALL Op Code

K-series routines can be accessed through use of the standard subroutine linkage mechanism and the CALL op code. The format of this procedure is as follows:

```
          .PSECT  code
          MOV     #arglist,R5     ;ARGUMENT ADDRESS TO R5
          CALL    ksubr           ;CALL K-SERIES ROUTINE
          .PSECT  data
arglist:  .BYTE   narg,0          ;NUMBER OF ARGUMENTS
          .WORD   addr1           ;FIRST ARGUMENT ADDRESS
    .
    .
    .
          .WORD   addrn           ;LAST ARGUMENT ADDRESS
```

In this sample, the two PSECT directives are shown only to indicate the noncontiguity of the code and data portions of the linkage mechanism. Within the argument list, any argument that is to be defaulted must be represented by a -1 (that is, 177777_8).

13.2.2.2 Special-Purpose Macros

To facilitate the calling of K-series support routines from a MACRO–11 program, two macros are provided in file [45,10]LABMAC.MAC. These macros are as follows:

INITS	Specifies INITS is an initialization macro. It should be invoked at the beginning of the MACRO–11 source module.
CALLS	Invokes a K-series support routine.

Format

CALLS ksubr,ARG1,...,ARGN

Parameters

ksubr

Specifies the name of a K-series support routine.

arg1,...,argn

Specifies arguments to be formatted into an argument list and to be passed to the routine. Each argument can be either a symbolic name or a constant (interpreted as a positive decimal number), or the argument can be defaulted.

13.2.3 The I/O Status Block

Each active sweep must have its own I/O status block (IOSB). The IOSB is a 2-word array allocated in your task. It receives the status of a call to a K-series support routine. When a data sweep routine is called, the IOSB is always the first two words of the 40-word array specified as the first argument of the call. The first word of the IOSB contains the status code. The second word contains the buffer size in words.

The codes that can appear in the first word of an IOSB are in ISA-compatible format (with the exception of the I/O pending condition). Table 13–3 lists all return codes.

Table 13–3: Contents of First Word of IOSB

IOSB Word 1	Meaning
0	Operation pending; I/O in progress
1	Successful completion
301	Invalid arguments
305	Hardware or software option not present
306	Illegal buffer specification
313	Data overrun
315	Request terminated
317	Resource in use
397	Invalid event flag

13.3 Buffer Management

The management of buffers for data sweeps by K-series support routines involves the use of the following two FIFO (first-in/first-out) queues:

- The device queue (DVQ)
- The user task queue (USQ)

The device queue (DVQ) contains the numbers of all buffers that your task has released to the support routines in a call to RLSBUF. The buffers represented by these numbers are ready to be filled with data (input sweeps) or to be emptied of data (output sweeps). Any buffer specified in a call to INXTBF must already be in DVQ.

The user task queue (USQ) contains the numbers of buffers available to your task. For output sweeps, this queue contains the numbers of buffers that have already been emptied by the driver. For input sweeps, the buffers represented by USQ are those buffers that are filled with data. In both instances, your task determines the next buffer to use (that is, extracts the first element of USQ) by calling IGTBUF or IWTBUF.

Both the DVQ and USQ are initialized to -1, indicating no buffers, when your task calls the SETIBF routine. The task must call RLSBUF before initiating any sweep because at least one buffer must be present in DVQ for the first input or output to occur.

For input sweeps, the best strategy is to call RLSBUF and specify the numbers associated with all the buffers to be used in the sweep.

For output sweeps, one approach is to specify two buffers (for continuous sweeps) in the call to RLSBUF. The first action then taken either in a completion routine or after a call to IWTBUF would be to release the next buffer. Note, however, that this approach does not represent true multiple buffering because data overrun occurs if the second buffer is not released in time.

13.4 Sample FORTRAN Programs

Two sample FORTRAN programs showing the use of K-series support routines are presented in this section. The first program uses event flags for internal synchronization. The second program demonstrates the use of a completion routine that you supply for synchronization.

Note

FORTRAN completion routines must not contain any of the following:

- Any I/O through the FORTRAN run-time system
- Any use of virtual arrays
- Any use of floating-point operations
- Any errors (error reporting is done through the FORTRAN run-time system)
- Anything else that may change the FORTRAN impure area

Any of the above may result in fatal task errors or unpredictable results.

13.4.1 Sample Program Using Event Flag

The following example illustrates the use of an event flag within a program.

```
      IMPLICIT INTEGER (A-Z)
      DIMENSION BUF(1024,8), IBUF (40), IOSB(2)
      EQUIVALENCE (IBUF(1),IOSB(1))
C
C      INITIALIZE THE IBUF ARRAY FOR THE A/D SWEEP
C
      CALL SETIBF (IBUF,IND, , BUF(1,1), BUF(1,2), BUF(1,3),
     * BUF(1,4), BUF(1,5), BUF(1,6), BUF(1,7), BUF(1,8))
      WRITE (1, 900)
      READ (1, 910) IRATE, IPRSET
C
C      SET THE CLOCK RATE AND PRESET FOR THE SWEEP
C
      CALL CLOCKA (IRATE, IPRSET,IND)
C
C      THIS IS INPUT, SO RELEASE ALL BUFFERS TO SERVICE
C      ROUTINE
C
      CALL RLSBUF (IBUF,IND, 0,1,2,3,4,5,6,7)
C
C      START THE SWEEP. USE 1024 WORD BUFFERS, SAMPLE
C      FOREVER, EXTERNAL START, EVENT FLAG 30, 1 CHANNEL (0).
C
      CALL ADSWP (IBUF, 1024, -1, 256, IPRSET,
     * 30, 0, 0, 1)
C
C      HERE WE COULD CHECK THE I/O STATUS BLOCK TO ENSURE
C      THAT THE SWEEP IS ACTUALLY RUNNING.
C
      IBFCNT=0
C
C      THIS IS THE TOP OF THE DATA PROCESSING LOOP. WE
C      WAIT FOR A BUFFER TO BE COMPLETED, AND THEN DUMP
C      THE FIRST 100 WORDS OF THE BUFFER TO LUN 1.
C
 10   IBUFNO = IWTBUF(IBUF, 30)+1
C
C      IWTBUF RETURNS A POSITIVE BUFFER NUMBER
C      AS LONG AS THERE IS A BUFFER OF DATA AVAILABLE.
C      IF IND IS -1, WE PROBABLY HAD DATA OVERRUN, SO STOP.
C
      IF (IBUFNO .EQ. 0) STOP
      IBFCNT=IBFCNT+1
      WRITE (1,920) IBFCNT
      WRITE (1,930) (BUF(I,IBUFNO), I=1,100)
```

```
C
C          RELEASE BUFFER FOR SERVICE ROUTINE TO REFILL
C
          CALL RLSBUF(IBUF,IND,IBUFNO-1)
          GOTO 10
 900      FORMAT (' ENTER IRATE, IPRSET:', $)
 910      FORMAT (I, 0)
 920      FORMAT (' DUMP OF BUFFER NUMBER ',I5,/)
 930      FORMAT (1X,1007)
          END
```

13.4.2 Sample Program Using Completion Routine

The following example illustrates the use of a completion routine in a program.

```
          IMPLICIT INTEGER (A-Z)
          EXTERNAL AST

          DIMENSION BUF(1024,8), IBUF (40), IOSB(2)
          COMMON /KDATA/ BUF, IBUF, IBFCNT
          EQUIVALENCE (IBUF(1),IOSB(1))

C
C            INITIALIZE THE IBUF ARRAY FOR THE A/D SWEEP
C
          CALL SETIBF (IBUF,IND, , BUF(1,1), BUF(1,2), BUF(1,3),
          * BUF(1,4), BUF(1,5), BUF(1,6), BUF(1,7), BUF(1,8))

          WRITE (1, 900)
          READ (1, 910) IRATE, IPRSET

C
C            SET THE CLOCK RATE AND PRESET FOR THE SWEEP
C
          CALL CLOCKA (IRATE, IPRSET,IND)

C
C            THIS IS INPUT, SO RELEASE ALL BUFFERS TO SERVICE
C            ROUTINE
C
          CALL RLSBUF (IBUF,IND, 0, 1, 2, 3, 4, 5, 6, 7)

C
C            START THE SWEEP. USE 1024 WORD BUFFERS, SAMPLE
C            FOREVER, EXTERNAL START, EVENT FLAG 30, 1 CHANNEL (0).
C
          IBFCNT = 0
          CALL ADSWP (IBUF, 1024, 0, 256, IPRSET
          * AST, 0, 0, 1)
```

```
C
C           HERE WE COULD CHECK THE I/O STATUS BLOCK TO ENSURE
C           THAT THE SWEEP IS ACTUALLY RUNNING.
C
 10       CALL WAITFR (23)
C
C           WHEN EVENT FLAG 23 IS SET THE SWEEP IS COMPLETED.
C           WE MAY EXIT NOW.
C
          STOP

 900      FORMAT (' ENTER IRATE, IPRSET:', $)
 910      FORMAT (I, O)
          END
          SUBROUTINE AST
C
C           THIS SUBROUTINE IS CALLED AT AST LEVEL WHENEVER
C           A BUFFER IS COMPLETED. THIS ROUTINE PROCESSES
C           THE CONTENTS OF THE BUFFER AND THEN RELEASES
C           IT FOR THE SERVICE ROUTINE. IF THE SWEEP IS TO
C           TERMINATE (IOSB NON-ZERO) THEN EVENT FLAG 23. IS
C           SET TO INDICATE TO THE MAINLINE CODE THAT WE ARE
C           DONE.
          IMPLICIT INTEGER (A-Z)
          DIMENSION BUF(1024,8), IBUF(40), IOSB(2)
          COMMON /KDATA/ BUF, IBUF, IBFCNT
          EQUIVALENCE (IBUF(1),IOSB(1))

          IBUFNO = IGTBUF (IBUF) +1

          IF (IBUFNO-1) .GE. 0 GOTO 20
          IF (IOSB(1) .EQ. 0) PAUSE 'INCONSISTENT STATE'
          CALL SETEF (23)
          RETURN

 20       IBFCNT = IBFCNT + 1

C
C           HERE WE WOULD PROCESS THE DATA
C
C
C           RELEASE BUFFER FOR SERVICE ROUTINE
C
          CALL RLSBUF (IBUF, IND, IBUFNO-1)
          RETURN

          END
```

Chapter 14

UNIBUS Switch Driver

14.1 Introduction to the UNIBUS Switch Driver

The UNIBUS switch driver supports DT07 UNIBUS switch hardware on RSX-11M-PLUS systems. UNIBUS switches are electronic devices that allow peripherals to be switched from one central processing unit (CPU) to another, enabling CPUs to share peripheral devices. UNIBUS switches also facilitate online system backup and allow dynamic reconfiguration of systems in which high availability of certain peripherals is required.

14.1.1 DT07 UNIBUS Switches

DT07 UNIBUS switches can provide two, three, or four ports for connecting an external UNIBUS run to one of two, three, or four CPUs.

Any CPU can request connection to a UNIBUS run and receive the connection immediately if the requested UNIBUS run is in the neutral state (it is not connected to another CPU's UNIBUS). If the request is received when the UNIBUS run is connected to another CPU, an interrupt is generated, informing the connected CPU of the pending request, and a watchdog timer is started. The connected CPU normally acknowledges the request, indicating the UNIBUS is still in use. In this case, the UNIBUS remains connected to the CPU. However, if the CPU does not respond to the interrupt within the time limit imposed by the DT07's watchdog timer, the UNIBUS is switched to the requesting CPU. Thus, a CPU that is not operating remains connected to the UNIBUS only until another CPU requests the UNIBUS.

Each DT07 UNIBUS switch port functions as an isolation circuit. When its power is off, it does not affect any CPU operation.

14.1.2 UNIBUS Switch Driver

The UNIBUS switch driver allows you to use the UNIBUS switch in one of the following two ways:

- A CPU retains the UNIBUS until the task issuing the directives that connected the UNIBUS to this CPU exits. This is normally accomplished when the task attaches the UNIBUS switch (IO.ATT function) and issues the connect function (IO.CON). When the task exits (for any reason), the system detaches the UNIBUS switch (IO.DET) and performs an implicit disconnect function (IO.DIS), releasing the UNIBUS switch for use by any other task.

  The task that attaches the UNIBUS switch can be considered the manager of the UNIBUS switch until the task exits. The task can receive asynchronous system traps (ASTs) for certain conditions involving UNIBUS switching (see Section 14.3.1.1).

- A CPU retains the UNIBUS until a task is executed that explicitly disconnects the UNIBUS. This is normally accomplished when a task issues the IO.CON function and no previous IO.ATT was issued. Once the UNIBUS is connected, the task exits. The UNIBUS then remains connected until either the CPU fails to respond to other CPU requests for the UNIBUS, or a task is executed that explicitly disconnects the UNIBUS. Note that when operating in this manner, no active task is required to retain the UNIBUS.

14.2 Get LUN Information Macro

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains all zeros. Words 3, 4, and 5 are undefined.

14.3 QIO$ Macro

This section summarizes standard and device-specific QIO functions for UNIBUS switches.

14.3.1 Standard QIO Functions

Table 14–1 lists the standard functions of the QIO macro that are valid for UNIBUS switches.

Table 14–1: Standard QIO Functions for UNIBUS Switches

Format	Function
QIO$C IO.ATT,..., <[ast]>	Attach device
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Cancel I/O requests

Parameter

ast

Specifies the address of an optional AST routine that is entered if certain conditions are detected (see Section 14.3.1.1).

IO.ATT does not connect the UNIBUS switch (see device-specific function IO.CON).

IO.DET detaches the UNIBUS switch from the task. If the UNIBUS switch was previously attached by the IO.CON function, an implied disconnect (IO.DIS) function is performed.

The only I/O requests that can be affected by the IO.KIL function are IO.CON and IO.DPT. When IO.KIL is issued during an IO.CON function, further retries are canceled. When IO.KIL is issued during an IO.DPT function, the timeout count is changed, forcing timeout (IE.TMO) to occur.

14.3.1.1 IO.ATT

The IO.ATT QIO function attaches the UNIBUS switch to the task issuing the QIO directive. An optional AST address parameter can be specified. However, if it is specified, it must remain valid while the UNIBUS switch remains attached to the task.

The AST service routine for the UNIBUS switch is entered when one of the following conditions occur:

- The UNIBUS switch has become connected to another CPU due to one of the following conditions:
  - The operator manually switched the UNIBUS to another CPU.
  - This CPU failed to respond to another CPU's request for the UNIBUS within the specified time (the CPU must acknowledge the request by servicing an interrupt, as described in Section 14.1.1).

  UNIBUS switch condition code 1 is passed to the AST routine by the stack, indicating the cause of the AST.
- The UNIBUS switch has disconnected from the CPU due to one of the following conditions:
  - A power failure occurred in this CPU (system power failure) and the UNIBUS switch driver was unable to reconnect the bus
  - A power failure occurred on the connected UNIBUS, causing the driver to disconnect the UNIBUS

  UNIBUS switch condition code 2 (for a system power failure) or condition code 3 (for a UNIBUS power failure) is passed to the AST routine by the stack, indicating the cause of the AST.

14.3.1.2 IO.DET

The IO.DET function detaches the issuing task from the UNIBUS switch, and, in addition, performs an implied disconnect for the issuing task if that task had connected the UNIBUS switch. A detach function is generated by the Executive on behalf of an attached task if that task exits (normally or abnormally) without explicitly detaching the device. For a switched UNIBUS, this causes the UNIBUS to be disconnected if an attached connected task faults in such a way as to cause the task to exit.

14.3.1.3 IO.KIL

The IO.KIL function cancels any outstanding IO.CON function that has a nonzero retry count and any outstanding IO.DPT function that has not yet timed out. Other QIO functions in progress are not affected by IO.KIL, and they are completed.

14.3.2 Device-Specific QIO Functions

The device-specific functions of the QIO macro that are valid for UNIBUS switches are shown in Table 14–2.

Table 14–2: Device-Specific QIO Functions for UNIBUS Switches

Format	Function
QIO$C IO.CON,..., <[rcnt],[cpu]>	Connect UNIBUS switch
QIO$C IO.DIS,..., <[tout],[port]>	Disconnect UNIBUS switch
QIO$C IO.DPT,..., <[tout],[port]>	Disconnect UNIBUS switch from specified CPU port
QIO$C IO.SWI,..., <cpu>	Switch the UNIBUS from current CPU to specified CPU
QIO$C IO.CSR,...	Read UNIBUS switch control and status register (CSR)

Parameters

rcnt

Specifies the number of additional times the connect is attempted if the IO.CON fails to complete.

cpu

Specifies the American Standard Code for Information Interchange (ASCII) letter designating the CPU to receive the UNIBUS switch.

port

Specifies the port number, ranging from 0 to 3, of the target CPU that must request the bus prior to the CPU that is currently connected to the UNIBUS actually completing the disconnect. The port number corresponds to the four MANUAL CONNECT switch positions (PORT 0 to PORT 3) marked on the DT07 control panel.

tout

Specifies the maximum time (in seconds) allowed (253_{10} maximum) for the function to be completed before an error condition is reported.

Parameter details are included in the following sections.

14.3.2.1 IO.CON

The IO.CON (connect) function requests connection of a UNIBUS presently not connected to a specified CPU. It can be issued either by a task previously attached with the IO.ATT function or by a task that is not attached. The IO.CON function has four optional parameters. The use of each parameter is described as follows:

rcnt

The retry count (rcnt) specifies the number of additional times the connect function is attempted if the IO.CON fails to complete within the timeout period of the UNIBUS switch. Retry count parameters used in this manner are always nonzero positive values.

The IO.CON function is not completed until either the retry count expires or the UNIBUS switch is connected successfully. Thus, the issuing task having a nonzero retry count is not checkpointed until the IO.CON function is completed.

When a retry count of 0 is specified, the connect function attempts to connect the UNIBUS switch once (no retries) and immediately reports the directive status to the issuing task.

When a retry count of 177,777 (-1) is specified, the connect function continues to retry the connection until a successful connection is made or an IO.KIL function is issued.

cpu

You can use the CPU parameter only with loosely coupled multiprocessor systems to specify the CPU to which the UNIBUS switch should be connected. A loosely coupled system is one in which memory resources are not shared by more than one CPU. Use this function only when the UNIBUS switch is presently not connected (you should use the IO.SWI function to disconnect the UNIBUS switch from a connected closely coupled CPU and to connect it to a specified closely coupled CPU). Specify the CPU by a single ASCII letter (A, B, C, or D).

14.3.2.2 IO.DIS

The IO.DIS function disconnects the switched UNIBUS from the currently connected CPU. Note if your task issues the IO.DIS or IO.DPT function, it must determine that all devices on the switched UNIBUS are inactive when it issues the function. The UNIBUS switch driver does not check for active devices on the UNIBUS before completing either the IO.DIS or IO.DPT function.

14.3.2.3 IO.DPT

Use the IO.DPT function in a loosely coupled multiprocessor system to allow the UNIBUS to be connected to another CPU on a specified port if the CPU requests connection within a specified time interval. A loosely coupled system is one in which memory resources are not shared

by more than one CPU. The IO.DPT function has two optional parameters. The use of each parameter is described as follows:

tmo — The timeout parameter specifies the maximum time allowed for the function to complete before an error is reported. Timeout specifications are positive, nonzero values ranging from 1 to 254 seconds. The default timeout value is 2 seconds. If the CPU parameter is included in the IO.DPT function, the driver waits for the specified CPU to request the UNIBUS up to the specified timeout value. If the CPU does not request the UNIBUS during this time, the UNIBUS remains connected and the IE.TMO status is returned to the issuing task.

If a timeout value of 0 is specified, the IO.DIS function does not complete until either the successful disconnect occurs or an IO.KIL function is issued.

port — You can use the port parameter only with loosely coupled multiprocessor systems to specify the port through which the UNIBUS switch should be connected to a CPU. Specify the port by a number ranging from 0 to 3.

14.3.2.4 IO.SWI

The IO.SWI function disconnects the UNIBUS switch from the currently connected CPU and connects it to the specified CPU in a closely coupled system. The cpu parameter is required.

IO.SWI is executed without the possibility of a third CPU taking control of the UNIBUS during the switching process.

Use the CPU parameter in closely coupled multiprocessor systems to specify the CPU to which the UNIBUS switch should be connected. Specify the CPU by a single ASCII letter (A, B, C, or D).

14.3.2.5 IO.CSR

The IO.CSR function reads maintenance information contained in the device CSR and returns it in the second word of the I/O status block (IOSB). Information returned is valid only if the UNIBUS switch is connected. Limit the use of this function to diagnostic applications.

14.4 Powerfail Recovery

The following sections describe what action the UNIBUS switch driver takes when a power failure occurs.

14.4.1 System Powerfail Recovery

During powerfail recovery, the driver attempts to restore the state of the system prior to the actual power failure. If the UNIBUS switch is found to be disconnected during powerfail recovery, the driver attempts to reconnect the switched UNIBUS. If the first attempt to reconnect the UNIBUS is not successful, an entry is made in the error log and the attached task is notified of the UNIBUS switch state by the AST specified in the IO.ATT function (if previously issued).

If an IO.CON function was in progress when the power failure occurred and a retry count was pending, the UNIBUS switch driver attempts to successfully connect the UNIBUS switch until the retry count expires.

If an IO.DIS or IO.DPT function was in progress when the power failure occurred, the UNIBUS switch driver attempts to complete the operation.

14.4.2 UNIBUS Powerfail Recovery

If an interrupt is received from the UNIBUS switch indicating a power failure has occurred on the switched UNIBUS, the driver issues an immediate disconnect (IO.DIS). The attached task (if any) is notified by the AST. Note that the system may be corrupted if some of the I/O devices on the switched UNIBUS were active when the power failure occurred because the drivers for those I/O devices may attempt to access the device registers after the switched UNIBUS (and I/O devices) has become disconnected.

14.5 Status Returns

Table 14–3 lists the error and status conditions that are returned by the UNIBUS switch driver.

Table 14–3: UNIBUS Switch Driver Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO directive was completed successfully.
IS.PND	**I/O request pending** The operation specified in the QIO directive has not been executed yet. The IOSB is filled with zeros.
IE.ABO	**Request aborted** An I/O request was queued (not yet acted upon by the driver) when an IO.KIL was issued.
IE.BAD	**Bad parameters** The parameters specified in the QIO macro were in error.
IE.CNR	**Connect rejected** The connect function did not successfully connect the switched UNIBUS to the specified CPU, and the retry count, if specified, has expired.
IE.DAA	**Device already attached** The device specified in an IO.ATT function was already attached by the issuing task. This code indicates that the issuing task has already attached the desired physical device unit, not that the unit was attached by another task.
IE.DNA	**Device not attached** The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.
IE.IFC	**Illegal function code** A function code was specified in an I/O request that is illegal for the UNIBUS switch driver.
IE.NOD	**Insufficient buffer space** Dynamic storage space has been depleted, resulting in insufficient buffer space available to allocate either the I/O packet or the device list buffer.

Table 14–3 (Cont.): UNIBUS Switch Driver Status Returns

Code	Reason
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO directive (the UNIBUS switch) was not on line, or the CPU specified in the IO.CON or IO.SWI was not on line.
IE.SPC	**Illegal address space** The buffer specified in the IO.CON function was partially or totally outside the address space of the issuing task.
IE.TMO	**Timeout error** The timeout count expired during an IO.DPT operation before the target CPU requested the UNIBUS. This error code is also returned when the DT03/DT07 hardware fails to respond to a request due to a hardware failure.

14.6 FORTRAN Usage

FORTRAN tasks can use all of the QIO functions described for the UNIBUS switch driver, except AST support is not provided (IO.ATT function with an AST address specified). You can write a macro subroutine, which the FORTRAN task can call, that specifies the AST address.

Appendix A
Summary of I/O Functions

This appendix summarizes valid I/O functions for all device drivers described in this manual. Both devices and functions are listed alphabetically. The meanings of the five parameters represented by the ellipsis (. . .) are described in Chapter 1. The meanings of the function-specific parameters shown below are discussed in the appropriate driver chapters. The user may reference these functions symbolically by invoking the system macros FILIO$ (standard I/O functions) and SPCIO$ (special I/O functions), or by allowing them to be defined at task-build time from the system object library.

A.1 Card Reader Driver

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O requests
IO.RDB,..., <stadd,size>	Read logical block (binary)
IO.RLB,..., <stadd,size>	Read logical block (alphanumeric)
IO.RVB,..., <stadd,size>	Read virtual block (alphanumeric)

A.2 DECtape II DRIVER

IO.ATT...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O requests
IO.RLB,..., <stadd,size,,,lbn>	Read logical block
IO.WLB,..., <stadd,size,,,lbn>	Write logical block
IO.WLC,..., <stadd,size,,,lbn>	Write logical block with check

IO.RLC,..., <stadd,size,,,lbn>	Read logical block with check
IO.BLS,..., <lbn>	Position tape
IO.DGN,...	Run internal diagnostics

A.3 DEUNA Driver

IO.XOP,..., <p1,p2,p3>	Open a line
IO.XSC,..., <p1,p2>	Set characteristics (Ethernet)
IO.XIN,..., <p1>	Initialize the line
IO.XRC,..., <p1,p2,p3,p4,[p5,p6]>	Receive a message on the line
IO.XTM,..., <p1,p2,p3,p4,[p5,p6]>	Transmit a message on the line
IO.XCL,...	Close the line
IO.XTL+subfunction,...	Control function

A.4 Disk Driver

IO.RLB,..., <stadd,size,,blkh,blkl>	Read logical block
IO.RPB,..., <stadd,size,,,pbn>	Read physical block
IO.RVB,..., <stadd,size,,blkh,blkl>	Read virtual block
IO.SEC,..., <stadd,size,pbn>	Sense characteristics (RX02 only)
IO.SMD,..., <density,,>	Set media density (RX02 only)
IO.WDD,..., <stadd,size,,,pbn>	Write physical block (with deleted data mark)
IO.WLB,..., <stadd,size,,blkh,blkl>	Write logical block
IO.WLC,..., <stadd,size,,blkh,blkl>	Write logical block followed by write-check
IO.WPB,..., <stadd,size,,,pbn>	Write physical block
IO.WVB,..., <stadd,size,,blkh,blkl>	Write virtual block

A.5 Laboratory Peripheral Accelerator Driver

IO.CLK,..., <mode,ckcsr,preset>	Start clock
IO.INI,..., <irbuf,278.>	Initialize LPA11-K
IO.LOD,..., <mbuf,2048.>	Load microcode
IO.STA,..., <bufptr,40.>	Start data transfer
IO.STP,..., <userid>	Stop request

A.6 Line Printer Driver

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O requests
IO.WLB,..., <stadd,size,vfc>	Write logical block
IO.WVB,..., <stadd,size,vfc>	Write virtual block

A.7 Magnetic Tape Driver

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.DSE,...	Data security erase (TK50/TU81 only)
IO.EOF,...	Write end-of-file (EOF)(tape mark)
IO.ERS,...	Erase (TE10 and TU10 not supported)
IO.KIL,...	Cancel I/O requests
IO.RLB,..., <stadd,size>	Read logical block
IO.RLV,..., <stadd,size>	Read logical block reverse
IO.RVB,..., <stadd,size>	Read virtual block
IO.RWD,...	Rewind tape
IO.RWU,...	Rewind and turn unit off line
IO.SEC,...	Read tape characteristics
IO.SMO,..., <cb>	Mount tape and set tape characteristics
IO.SPB,..., <nbs>	Space blocks
IO.SPF,..., <nes>	Space files
IO.STC,..., <cb>	Set tape characteristics
IO.WLB,..., <stadd,size>	Write logical block
IO.WVB,..., <stadd,size>	Write virtual block

A.8 Parallel Communication Link Drivers

Transmitter Driver Functions

IO.ATX,..., <stadd,size,flagwd,id,retries,retadd>	Attempt message transmission
IO.STC,..., <stadd,size,[state],[mode],,retadd>	Set master section characteristics
IO.SEC,...,	Sense master section status

Receiver Driver Functions

IO.CRX,..., <tef>	Correct for reception
IO.ATF,..., <stadd,size,retadd>	Accept transfer
IO.RTF,...	Reject transfer
IO.DRX,...	Disconnect from reception

A.9 Terminal Driver

IO.ATA,..., <ast,[parameter2],[ast2]>	Attach device, specify unsolicited-character asynchronous system trap (AST)
IO.ATT,...	Attach device
IO.CCO,..., <stadd,size,vfc>	Write logical block, cancel CTRL/O
IO.DET,...	Detach device
IO.EIO!TF.RLB,..., <stadd,size>	Extended I/O read functions
IO.EIO!TF.WLB,..., <stadd,size>	Extended I/O write functions
IO.GTS,..., <stadd,size>	Get terminal support
IO.HNG,...	Hang up remote line
IO.KIL,...	Cancel I/O requests
IO.RAL,..., <stadd,size,[tmo]>	Read logical block and pass all characters
IO.RLB,..., <stadd,size,[tmo]>	Read logical block
IO.RNE,..., <stadd,size,[tmo]>	Read logical block and do not echo
IO.RPR,..., <stadd,size,[tmo],pradd,prsize,vfc>	Read after prompt
IO.RST,..., <stadd,size,[tmo]>	Read with special terminators
IO.RTT,..., <stadd,size,[tmo],table>	Read logical block ended by specified special terminators
IO.RVB,..., <stadd,size,[tmo]>	Read virtual block

IO.WAL,..., <stadd,size,vfc>	Write logical block and pass all characters
IO.WBT,..., <stadd,size,vfc>	Write logical block and break through any ongoing I/O
IO.WLB,..., <stadd,size,vfc>	Write logical block
IO.WVB,..., <stadd,size,vfc>	Write virtual block
SF.GMC,..., <stadd,size>	Get multiple characteristics
SF.SMC,..., <stadd,size>	Set multiple characteristics

Subfunction bits for terminal-driver functions:

TF.AST	Unsolicited-input-character AST
TF.BIN	Binary prompt
TF.CCO	Cancel CTRL/O
TF.ESQ	Recognize escape sequences
TF.NOT	Unsolicited input AST notification
TF.RAL	Read, pass all characters
TF.RCU	Restore cursor position
TF.RDI	Read with default input (IO.EIO function only)
TF.RES	Read with escape sequence processsing enabled (IO.EIO function only)
TF.RLB	Read logical block (IO.EIO function only)
TF.RLU	Read and convert from lowercase to uppercase (IO.EIO function only)
TF.RNE	Read with no echo
TF.RNF	Read with no filter (IO.EIO function only)
TF.RPR	Read after prompt (IO.EIO function only)
TF.RPT	Read in pass-through mode (IO.EIO function only)
TF.RST	Read with special terminators
TF.RTT	Read with specified special terminator table (IO.EIO function only)
TF.TMO	Read with timeout
TF.WAL	Write, pass all bits
TF.WBT	Breakthrough write

TF.WIR	Write with input redisplayed
TF.XCC	CTRL/C starts a command line interpreter (CLI)
TF.XOF	Send XOFF

A.10 UNIBUS Switch Driver

IO.ATT,..., <[ast]>	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O requests
IO.CON,..., <[rcnt],[cpu]>	UNIBUS switch
QIO$C IO.DIS,...,	Disconnect UNIBUS switch
IO.DPT,..., <[tout],[port]>	Disconnect UNIBUS switch and connect to specified central processing unit (CPU) port
IO.SWI,..., <cpu>	Switch UNIBUS from current CPU to specified CPU
IO.CSR,...	Read UNIBUS switch CSR

A.11 Virtual Terminal Driver

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.KIL,...	Cancel I/O request
IO.RLB,..., <stadd,size>	Read logical block
IO.RVB,..., <stadd,size>	Read virtual block
IO.WLB,..., <stadd,size,stat>	Write logical block
IO.WVB,..., <stadd,size,stat>	Write virtual block
IO.STC,..., <cb,sw2,sw1>	Set terminal characteristics (enable/ disable intermediate buffering, or return I/O completion status)

Appendix B
I/O Function and Status Codes

This appendix lists the numeric codes for all I/O functions, directive status returns, and I/O completion status returns. The lists are organized in the following sequence:

- I/O completion status codes
- Directive status codes
- Device-independent I/O function codes
- Device-dependent I/O function codes

Device-dependent function codes are listed by device. Both devices and codes are organized in alphabetical order.

For each code, the symbolic name is listed in form IO.xxx, IE.xxx, or IS.xxx. A brief description of the error or function is also included. Both decimal and octal values are provided for all codes.

B.1 I/O Completion Status Codes

This section lists error and success codes that can be returned in the IOSB upon completion of an I/O function. The codes shown in the following sections may be referenced symbolically by invoking the system macro IOERR$.

B.1.1 I/O Error Status Codes

Name	Decimal	Octal	Meaning
IE.2DV	-48	177720	Rename—two different devices
IE.ABO	-15	177761	Operation aborted
IE.ALC	-84	177654	Allocation failure

Name	Decimal	Octal	Meaning
IE.ALN	-34	177736	File already accessed on logical unit number (LUN)
IE.BAD	-01	177777	Bad parameter
IE.BBE	-56	177710	Bad block on device
IE.BCC	-66	177676	Block check error or framing error
IE.BDI	-52	177714	Bad directory syntax
IE.BDR	-50	177716	Bad directory file
IE.BDV	-55	177711	Bad device name
IE BHD	-64	177700	Bad file header
IE.BLB	-70	177672	Bad logical buffer
IE.BLK	-20	177754	Invalid block number; logical block number too large
IE.BNM	-54	177712	Bad file name
IE.BTF	-76	177664	Bad tape format
IE.BTP	-43	177725	Bad record type
IE.BVR	-63	177701	Bad version number
IE.BYT	-19	177755	Odd byte count (or virtual address); byte-aligned buffer specified
IE.CKS	-30	177742	File header checksum failure
IE.CLO	-38	177732	File was not closed properly
IE.CNR	-96	177640	Connection rejected
IE.CON	-22	177752	Universal Digital Controller (UDC) connect error
IE.DAA	-08	177770	Device already attached
IE.DAO	-13	177763	Data overrun
IE.DFU	-24	177750	Device full
IE.DIS	-69	177673	Path lost to partner
IE.DNA	-07	177771	Device not attached
IE.DNR	-03	177775	Device not ready
IE.DSQ	-90	177646	Disk quota exceeded
IE.DUN	-09	177767	Device not attachable
IE.DUP	-57	177707	Enter—duplicate entry in directory
IE.EOF	-10	177766	End-of-file encountered
IE.EOT	-62	177702	End-of-tape encountered

Name	Decimal	Octal	Meaning
IE.EOV	-11	177765	End-of-volume encountered
IE.EXP	-75	177665	File expiration date not reached
IE.FEX	-49	177717	Rename a new file name already in use
IE.FHE	-59	177705	Fatal hardware error
IE.FLG	-89	177647	Event flag already specified
IE.FLN	-81	177657	Device already off line
IE.FOP	-53	177713	File already open
IE.HFU	-28	177728	File header full
IE.ICE	-47	177721	Internal consistency error
IE.IES	-82	177656	Invalid escape sequence
IE.IFC	-02	177776	Invalid function code
IE.IFU	-25	177747	Index file full
IE.ILL	-42	177726	Invalid operation on File Descriptor Block (FDB)
IE.IQU	-91	177645	Inconsistent qualifier usage
IE.ISQ	-61	177703	Invalid sequential operation
IE.LCK	-27	177745	Locked from read/write access
IE.MII	-99	177635	Media inserted incorrectly
IE.MOD	-21	177753	Invalid UDC or ICS/ICR module
IE.NBF	-39	177731	No buffer space available for file
IE.NBK	-41	177727	File exceeds space allocated—no blocks
IE.NDA	-78	177662	No data available
IE.NDR	-72	177670	No dynamic space available
IE.NFI	-60	177704	File ID was not specified
IE.NFW	-69	177673	Path lost to partner
IE.NLK	-79	177661	Task not linked to specified ICS/ICR interrupts
IE.NLN	-37	177733	No file accessed on LUN
IE.NNC	-77	177663	Not American National Standards Institute (ANSI) D format byte count
IE.NNN	-68	177674	No such node
IE.NNT	-94	177642	Not a network task
IE.NOD	-23	177751	Caller's nodes exhausted; no dynamic memory available

Name	Decimal	Octal	Meaning
IE.NSF	-26	177746	No such file
IE.NST	-80	177660	Specified task not installed
IE.NRJ	-74	177666	Network connection reject
IE.NTR	-87	177651	Task not triggered
IE.OFL	-65	177677	Device off line
IE.ONL	-67	177675	Device on line
IE.ONP	-05	177773	Hardware option not present
IE.OVR	-18	177756	Invalid read overlay request
IE.PES	-83	177655	Partial escape sequence
IE.PRI	-16	177760	Privilege violation
IE.RAC	-44	177724	Invalid record access bits set
IE.RAT	-45	177723	Invalid record attribute bits set
IE.RBG	-40	177730	Invalid record size
IE.RCN	-46	177722	Invalid record number—too large
IE.REJ	-88	177650	Transfer rejected by receiving CPU
IE.RER	-32	177740	File processor device read error
IE.RES	-92	177644	Circuit reset during operation
IE.RNM	-51	177715	Cannot rename old file system
IE.RSU	-17	177757	Shareable resource in use
IE.SNC	-35	177735	File ID, file number check
IE.SPC	-06	177772	Invalid user buffer
IE.SPI	-100	177634	Spindown ignored
IE.SQC	-36	177734	File ID, sequence number check
IE.SRE	-14	177762	Send/receive failure
IE.STK	-58	177706	Not enough stack space File Control Services (FCS) or File Control Primitive (FCP)
IE.SZE	-98	177636	Unable to size device
IE.TML	-93	177643	Too many links to task
IE.TMO	-95	177641	Timeout on request
IE.UKN	-97	177637	Unknown name
IE.ULK	-85	177653	Unlock error

Name	Decimal	Octal	Meaning
IE.URJ	-73	177667	Connection rejected by user
IE.VER	-04	177774	Parity error on device
IE.WAC	-29	177743	Accessed for write
IE.WAT	-31	177741	Attribute control list format error
IE.WCK	-86	177652	Write-check error
IE.WER	-33	177737	File processor device write error
IE.WLK	-12	177764	Write-locked device

B.1.2 I/O Status Success Codes

Name	Code (Low Byte)	Subcode (High Byte)	Octal Word	Meaning
IS.CR	1	15	006401	Successful completion with carriage return
IS.CC	1	3	001401	Successful completion on read terminated by CTRL/C
IS.ESC	1	33	015401	Successful completion with ESCAPE
IS.ESQ	1	233	115401	Successful completion with an escape sequence
IS.PND	+00		000000	I/O request pending
IS.RDD	+02		000002	Deleted data mark read
IS.SUC	+01		000001	Successful completion
IS.TMO	+02		000002	Successful completion on read terminated by timeout
IS.TNC	+02		000002	Successful transfer but message truncated (receiver buffer too small)

B.2 Directive Status Codes

This section lists error and success codes that can be returned in the Directive Status Word (DSW) at symbolic location $DSW when a QIO directive is issued.

B.2.1 Directive Error Codes

Name	Decimal	Octal	Meaning
IE.ACT	-07	177771	Task not active
IE.ADP	-98	177636	Invalid address
IE.ALG	-84	177654	Alignment error
IE.AST	-80	177660	Directive issued/not issued from AST
IE.CKP	-10	177766	Issuing task not checkpointable
IE.FIX	-09	177767	Task already fixed/unfixed
IE.HWR	-06	177772	Device handler not resident
IE.IBS	-89	177647	Invalid send buffer size (.GT. 255_{10})
IE.IDU	-92	177644	Invalid device or unit
IE.IEF	-97	177637	Invalid event flag (.GT. 64_{10})
IE.ILU	-96	177640	Invalid LUN
IE.ILV	-19	177755	Invalid vector specified
IE.INS	-02	177776	Specified task not installed
IE.IOP	-83	177655	Window has I/O in progress
IE.IPR	-95	177641	Invalid priority (.GT. 250_{10})
IE.ITI	-93	177643	Invalid time parameters
IE.ITP	-88	177650	Invalid TI parameter
IE.ITS	-08	177770	Directive inconsistent with task state
IE.IUI	-91	177645	Invalid User Identification Code (UIC)
IE.LNL	-90	177646	LUN locked in use
IE.MAP	-81	177657	Invalid mapping specified
IE.NSW	-18	177756	No swap space available
IE.NVR	-86	177652	Invalid region ID
IE.NVW	-87	177651	Invalid address window ID
IE.PNS	-94	177642	Partition/region not in system
IE.PTS	-03	177775	Partition too small for task
IE.PRI	-16	177760	Privilege violation
IE.RBS	-15	177761	Receive buffer is too small
IE.RSU	-17	177757	Resource in use

Name	Decimal	Octal	Meaning
IE.SDP	-99	177635	Invalid Directive Identification Code (DIC) number or Directive Parameter Block (DPB) size
IE.TCH	-11	177765	Task is checkpointable
IE.ULN	-05	177773	Unassigned LUN
IE.UNS	-04	177774	Insufficient dynamic storage for send
IE.UPN	-01	177777	Insufficient dynamic storage
IE.WOV	-85	177653	Address window allocation overflow

B.2.2 Directive Success Codes

Name	Decimal	Octal	Meaning
IS.SUC	+01	000001	Directive accepted

B.3 I/O Function Codes

This section lists octal codes for all device-independent I/O functions. In addition, individual sections list octal codes for all device-dependent I/O functions.

B.3.1 Device-Independent I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.ATT	001400	3	0	Attach device
IO.DET	002000	4	0	Detach device
IO.KIL	000012	0	12	Cancel I/O requests
IO.RLB	001000	2	0	Read logical block
IO.RVB	010400	21	0	Read virtual block
IO.WLB	000400	1	0	Write logical block
IO.WVB	011000	22	0	Write virtual block

B.3.2 Specific DECtape II I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.WLC	000420	1	20	Write logical block with check
IO.RLC	001020	2	20	Read logical block with check
IO.BLS	004010	10	10	Position tape
IO.DGN	004150	10	150	Run internal diagnostics

B.3.3 Specific Disk I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.RPB	001040	2	40	Read physical block (RX01, RL01, and RL02 only)
IO.SEC				Sense characteristics (RX02 only)
	000000	0	0	Single Density
	040000	100	0	Double Density
IO.SMD	002510	5	110	Set media density (RX02/RX33 only)
IO.WDD	000540	1	140	Write physical block with deleted data mark (RX02 only)
IO.WLC	000420	1	20	Write logical block followed by write-check (all except RX01 and RX02)
IO.WPB	000440	1	40	Write physical block (RX01, RX02, RL01, and RL02 only)

B.3.4 Specific Magnetic Tape I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.DSE	003040	6	40	Data security erase (TK50/TU81 only)
IO.EOF	003000	6	0	Write end-of-file gap
IO.RLV	001100	2	100	Read logical block (reverse)

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.RWD	002400	5	0	Rewind tape
IO.RWU	002540	5	140	Rewind and unload
IO.SEC	002520	5	120	Sense characteristics
IO.SMO	002560	5	160	Mount and set characteristics
IO.SPB	002420	5	20	Space blocks
IO.SPF	002440	5	40	Space files
IO.STC	002500	5	100	Set characteristics

B.3.5 Specific Terminal I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.ATA	001410	3	10	Attach device, specify unsolicited-input-character AST
IO.CCO	000440	1	40	Write logical block and cancel CTRL/O
IO.EIO	017400	37	0	Extended I/O
IO.GTS	002400	5	00	Get terminal support
IO.HNG	003000	6	0	Hang up remote line
IO.RAL	001010	2	10	Read logical block and pass all bits
IO.RNE	001020	2	20	Read with no echo
IO.RPR	004400	11	00	Read after prompt
IO.RST	001001	2	1	Read with special terminators
IO.RTT	005001	12	1	Read logical block ended by specified special terminator (full-duplex driver only)
IO.WAL	000410	1	10	Write logical block and pass all bits
IO.WBT	000500	1	100	Write logical block and break through ongoing I/O
SF.GMC	002560	5	160	Get multiple characteristics
SF.SMC	002440	5	40	Set multiple characteristics

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
Subfunction Bits:				
	With IO.RLB, IO.RPR:			
	TF.RST	000001		
	TF.BIN	000002		
	TF.RAL	000010		
	TF.RNE	000020		
	TF.XOF	000100		
	TF.TMO	000200		
	With IO.WLB:			
	TF.RCU	000001		
	TF.WAL	000010		
	TF.CCO	000040		
	TF.WBT	000100		
	TF.WIR	000200		
	With IO.ATT:			
	TF.XCC	000001		
	TF.NOT	000002		
	TF.AST	000010		
	TF.ESQ	000020		
	With IO.EIO:			
	TF.WLB	000001		
	TF.RCU[1]	000001		
	TF.CCO[1]	000040		
	TF.WAL[1]	000010		
	TF.WBT[1]	000100		
	TF.WIR[1]	000200		

[1] Modifiers of the IO.EIO!TF.WLB subfunction. These are specified by you in the item-list buffer.

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
	TF.RLB	000002		
	TF.RLU[2]	000010		
	TF.RTT[2]	000400		
	TF.RST[2]	000001		
	TF.BIN[2]	000002		
	TF.RAL[2]	000010		
	TF.RNE[2]	000020		
	TF.XOF[2]	000100		
	TF.TMO[2]	000200		
	TF.RES[2]	010000		
	TF.RPR[2]	002000		
	TF.RPT[2]	004000		
	TF.RNF[2]	020000		
	TF.TNE[2]	040000		
	TF.RDI[2]	100000		

[2]Modifiers of the IO.EIO!TF.RLB subfunction. These are specified by you in the item-list buffer.

B.3.6 Specific Virtual Terminal I/O Function Codes

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.STC	002500	5	100	Set terminal characteristics

B.3.7 Specific A/D Converter I/O Function Codes—RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.RBC	003000	6	0	Initiate an analog-to-digital (A/D) conversion

B.3.8 Specific Card Reader I/O Function Codes—RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.RDB	001200	2	200	Read logical block (binary)

B.3.9 Specific Cassette I/O Function Codes—RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.EOF	003000	6	0	Write end-of-file gap
IO.RWD	002400	5	0	Rewind tape
IO.SPB	002420	5	20	Space blocks
IO.SPF	002440	5	40	Space files

B.3.10 Specific Communication (Message Oriented) I/O Function Codes—RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.FDX	003020	6	20	Set device to full-duplex mode
IO.HDX	003010	6	10	Set device to half-duplex mode
IO.INL	002400	5	0	Initialize device and set device characteristics
IO.RNS	001020	2	20	Read logical block, transparent mode
IO.SYN	003040	6	40	Specify sync character
IO.TRM	002410	5	10	Terminate communication, disconnecting from physical channel
IO.WNS	000420	1	20	Write logical block with no sync leader

B.3.11 Specific DECtape I/O Function Codes—RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.RLV	001100	2	100	Read logical block (reverse)
IO.WLV	000500	1	100	Write logical block (reverse)

B.3.12 Specific Parallel Communications Link I/O Function Codes—RSX-11M-PLUS Only

B.3.12.1 Transmitter Driver Functions

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.ATX	000400	1	0	Attempt message transmission
IO.STC	002500	5	100	Set master section characteristics
IO.SEC	002520	5	120	Sense master section status

B.3.12.2 Receiver Driver Functions

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CRX	014400	31	0	Connect for reception
IO.ATF	001000	2	0	Accept transfer
IO.RTF	015400	33	0	Reject transfer
IO.DRX	001500	32	0	Disconnect from reception

B.3.13 Specific UNIBUS Switch I/O Function Codes—RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CON	015400	33	0	Connect UNIBUS switch
IO.DIS	016000	34	0	Disconnect UNIBUS switch
IO.DPT	016010	34	10	Disconnect UNIBUS switch and connect to specified central processing unit (CPU) port
IO.SWI	016400	35	0	Switch UNIBUS from current CPU to specified CPU
IO.CSR	015000	32	0	Read UNIBUS switch control and status register (CSR)

Appendix C
Error Codes

This appendix includes the source code for the following:

- I/O error codes
- Directive Status Word (DSW) error codes
- I/O function codes

This source code is located in [61,10]QIOMAC.MAC.

```
        .TITLE      QIOMAC - QIOSYM MACRO DEFINITION
;
; DATE OF LAST MODIFICATION:
;
;       RYAN CHRISTOPHER     16-Nov-1984
;
;
; ***** ALWAYS UPDATE THE FOLLOWING TWO LINES TOGETHER
       .IDENT       /0375/
       QI.VER=0375
;
; COPYRIGHT (C) 1983, 1984
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN
; IN DEC.
;
```

```
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;
;
; SHANE MICHAEL   1-OCT-73
;
;+
; MACRO TO DEFINE STANDARD QUEUE I/O DIRECTIVE FUNCTION VALUES
; AND IOSB RETURN VALUES.  TO INVOKE AT ASSEMBLY TIME (WITH LOCAL
; DEFINITION) USE:
;
;       QIOSY$              ;DEFINE SYMBOLS
;
; TO OBTAIN GLOBAL DEFINITION OF THESE SYMBOLS USE:
;
;       QIOSY$ DEF$G        ;SYMBOLS DEFINED GLOBALLY
;
; THE MACRO CAN BE CALLED ONCE ONLY AND THEN
; REDEFINES ITSELF AS NULL.
;-

       .MACRO QIOSY$ $$$GBL,$$$MSG
       .IIF   IDN,<$$$GBL>,<DEF$G>, .GLOBL QI.VER
       .IF    IDN,<$$$MSG>,<DEF$S>
       $$$MAX=0
       $$MSG=1
       .IFF
       $$MSG=0
       .ENDC
       .MCALL   IOERR$
       IOERR$   $$$GBL               ;I/O ERROR CODES FROM HANDLERS, FCP, FCS
       .MCALL   DRERR$
       DRERR$   $$$GBL               ;DIRECTIVE STATUS WORD ERROR CODES
       .IF      DIF,<$$$MSG>,<DEF$S>
       .MCALL   FILIO$
       FILIO$   $$$GBL              ;DEFINE GENERAL I/O FUNCTION CODES
       .MCALL   SPCIO$
       SPCIO$   $$$GBL              ;DEVICE-DEPENDENT I/O FUNCTION CODES
       .MACRO   QIOSY$       ARG,ARG1,ARG2       ;RECLAIM MACRO STORAGE
       .ENDM    QIOSY$
       .ENDC
       .ENDM    QIOSY$
```

```
;
; DEFINE THE ERROR CODES RETURNED BY DEVICE HANDLER AND FILE PRIMITIVES
; IN THE FIRST WORD OF THE I/O STATUS BLOCK
; THESE CODES ARE ALSO RETURNED BY FILE CONTROL SERVICES (FCS) IN THE
; BYTE F.ERR IN THE FILE DESCRIPTOR BLOCK (FDB)
;       THE BYTE F.ERR+1 IS 0 IF F.ERR CONTAINS A HANDLER OR FCP ERROR CODE.
;
        .ENABL  LC
        .MACRO  IOERR$          $$$GBL
        .MCALL  .IOER.,DEFIN$
        .IF     IDN,<$$$GBL>,<DEF$G>
        ...GBL=1
        .IFF
        ...GBL=0
        .ENDC
        .IIF    NDF,$$MSG,$$MSG=0

;
; SYSTEM STANDARD CODES, USED BY EXECUTIVE AND DRIVERS
;

        .IOER.  IE.BAD,-01.,<Bad parameters>
        .IOER.  IE.IFC,-02.,<Invalid function code>
        .IOER.  IE.DNR,-03.,<Device not ready>
        .IOER.  IE.VER,-04.,<Parity error on device>
        .IOER.  IE.ONP,-05.,<Hardware option not present>
        .IOER.  IE.SPC,-06.,<Illegal user buffer>
        .IOER.  IE.DNA,-07.,<Device not attached>
        .IOER.  IE.DAA,-08.,<Device already attached>
        .IOER.  IE.DUN,-09.,<Device not attachable>
        .IOER.  IE.EOF,-10.,<End of file detected>
        .IOER.  IE.EOV,-11.,<End of volume detected>
        .IOER.  IE.WLK,-12.,<Write attempted to locked unit>
        .IOER.  IE.DAO,-13.,<Data overrun>
        .IOER.  IE.SRE,-14.,<Send/receive failure>
        .IOER.  IE.ABO,-15.,<Request terminated>
        .IOER.  IE.PRI,-16.,<Privilege violation>
        .IOER.  IE.RSU,-17.,<Shareable resource in use>
        .IOER.  IE.OVR,-18.,<Illegal overlay request>
        .IOER.  IE.BYT,-19.,<Odd byte count (or virtual address)>
        .IOER.  IE.BLK,-20.,<Logical block number too large>
        .IOER.  IE.MOD,-21.,<Invalid UDC module #>
        .IOER.  IE.CON,-22.,<UDC connect error>
        .IOER.  IE.BBE,-56.,<Bad block on device>
        .IOER.  IE.STK,-58.,<Not enough stack space (FCS or FCP)>
        .IOER.  IE.FHE,-59.,<Fatal hardware error on device>
        .IOER.  IE.EOT,-62.,<End of tape detected>
        .IOER.  IE.OFL,-65.,<Device off line>
        .IOER.  IE.BCC,-66.,<Block check, CRC, or framing error>
        .IOER.  IE.NFW,-69.,<Path lost to partner> ;THIS CODE MUST BE ODD
        .IOER.  IE.DIS,-69.,<Path lost to partner> ;DISCONNECTED (SAME AS NFW)
        .IOER.  IE.PNT,-71.,<Partition/Region not in system>
        .IOER.  IE.NDR,-72.,<No dynamic space available> ; SEE ALSO IE.UPN
        .IOER.  IE.TMO,-95.,<Timeout on request>        ; see also IS.TMO
        .IOER.  IE.CNR,-96.,<Connection rejected>
        .IOER.  IE.MII,-99.,<Media inserted incorrectly>
        .IOER.  IE.SPI,-100.,<Spindown ignored>
        .IOER.  IE.FER,-101.,<Forced error mark encountered>
```

```
;
; FILE PRIMITIVE CODES
;
        .IOER.    IE.NOD,-23.,<Caller's nodes exhausted>
        .IOER.    IE.DFU,-24.,<Device full>
        .IOER.    IE.IFU,-25.,<Index file full>
        .IOER.    IE.NSF,-26.,<No such file>
        .IOER.    IE.LCK,-27.,<Locked from read/write access>
        .IOER.    IE.HFU,-28.,<File header full>
        .IOER.    IE.WAC,-29.,<Accessed for write>
        .IOER.    IE.CKS,-30.,<File header checksum failure>
        .IOER.    IE.WAT,-31.,<Attribute control list format error>
        .IOER.    IE.RER,-32.,<File processor device read error>
        .IOER.    IE.WER,-33.,<File processor device write error>
        .IOER.    IE.ALN,-34.,<File already accessed on LUN>
        .IOER.    IE.SNC,-35.,<File ID, file number check>
        .IOER.    IE.SQC,-36.,<File ID, sequence number check>
        .IOER.    IE.NLN,-37.,<No file accessed on LUN>
        .IOER.    IE.CLO,-38.,<File was not properly closed>
        .IOER.    IE.DUP,-57.,<ENTER - duplicate entry in directory>
        .IOER.    IE.BVR,-63.,<Bad version number>
        .IOER.    IE.BHD,-64.,<Bad file header>
        .IOER.    IE.EXP,-75.,<File expiration date not reached>
        .IOER.    IE.BTF,-76.,<Bad tape format>
        .IOER.    IE.ALC,-84.,<Allocation failure>
        .IOER.    IE.ULK,-85.,<Unlock error>
        .IOER.    IE.WCK,-86.,<Write check failure>
        .IOER.    IE.DSQ,-90.,<Disk quota exceeded>
        .IOER.    IE.PIO,-104.,<Deaccessed failed due to pending I/O>

;
; FILE CONTROL SERVICES CODES
;
        .IOER.    IE.NBF,-39.,<symbol>(OPEN - no buffer space available for file)
        .IOER.    IE.RBG,-40.,<symbol>(Illegal record size)
        .IOER.    IE.NBK,-41.,<symbol>(File exceeds space allocated, no blocks)
        .IOER.    IE.ILL,-42.,<symbol>(Illegal operation on file descriptor block)
        .IOER.    IE.BTP,-43.,<symbol>(Bad record type)
        .IOER.    IE.RAC,-44.,<symbol>(Illegal record access bits set)
        .IOER.    IE.RAT,-45.,<symbol>(Illegal record attributes bits set)
        .IOER.    IE.RCN,-46.,<symbol>(Illegal record number - too large)
        .IOER.    IE.2DV,-48.,<symbol>(Rename - 2 different devices)
        .IOER.    IE.FEX,-49.,<symbol>(Rename - new file name already in use)
        .IOER.    IE.BDR,-50.,<symbol>(Bad directory file)
        .IOER.    IE.RNM,-51.,<symbol>(Can't rename old file system)
        .IOER.    IE.BDI,-52.,<symbol>(Bad directory syntax)
        .IOER.    IE.FOP,-53.,<symbol>(File already open)
        .IOER.    IE.BNM,-54.,<symbol>(Bad file name)
        .IOER.    IE.BDV,-55.,<symbol>(Bad device name)
        .IOER.    IE.NFI,-60.,<symbol>(File ID was not specified)
        .IOER.    IE.ISQ,-61.,<symbol>(Illegal sequential operation)
        .IOER.    IE.NNC,-77.,<symbol>(Not ANSI 'D' format byte count)

;
; NETWORK ACP, PSI, AND DECDATAWAY CODES
;
```

```
        .IOER.    IE.NNN,-68.,<No such node>
        .IOER.    IE.BLB,-70.,<Bad logical buffer>
        .IOER.    IE.URJ,-73.,<Connection rejected by user>
        .IOER.    IE.NRJ,-74.,<Connection rejected by network>
        .IOER.    IE.NDA,-78.,<No data available>
        .IOER.    IE.IQU,-91.,<Inconsistent qualifier usage>
        .IOER.    IE.RES,-92.,<Circuit reset during operation>
        .IOER.    IE.TML,-93.,<Too many links to task>
        .IOER.    IE.NNT,-94.,<Not a network task>
        .IOER.    IE.UKN,-97.,<Unknown name>
        .IOER.    IE.IRR,-102.,<Insufficient resources at remote node>
        .IOER.    IE.SUI,-103.,<Service in use>

;
; ICS/ICR ERROR CODES
;
        .IOER.    IE.NLK,-79.,<Task not linked to specified ICS/ICR interrupts>
        .IOER.    IE.NST,-80.,<Specified task not installed>
        .IOER.    IE.FLN,-81.,<Device offline when offline request was issued>

;
; TTY ERROR CODES
;
        .IOER.    IE.IES,-82.,<Invalid escape sequence>
        .IOER.    IE.PES,-83.,<Partial escape sequence>

;
; RECONFIGURATION CODES
;
        .IOER.    IE.ICE,-47.,<Internal consistency error>
        .IOER.    IE.ONL,-67.,<Device online>
        .IOER.    IE.SZE,-98.,<Unable to size device>

;
; PCL ERROR CODES
;
        .IOER.    IE.NTR,-87.,<Task not triggered>
        .IOER.    IE.REJ,-88.,<Transfer rejected by receiving CPU>
        .IOER.    IE.FLG,-89.,<Event flag already specified>
;
; SUCCESSFUL RETURN CODES---
;
        DEFIN$    IS.PND,+00.       ;OPERATION PENDING
        DEFIN$    IS.SUC,+01.       ;OPERATION COMPLETE, SUCCESS
        DEFIN$    IS.RDD,+02.       ;FLOPPY DISK SUCCESSFUL COMPLETION
                                    ;OF A READ PHYSICAL, AND DELETED
                                    ;DATA MARK WAS SEEN IN SECTOR HEADER
        DEFIN$    IS.TNC,+02.       ;(PCL) SUCCESSFUL TRANSFER BUT MESSAGE
                                    ;TRUNCATED (RECEIVE BUFFER TOO SMALL).
        DEFIN$    IS.CHW,+04.       ;(IBM COMM) DATA READ WAS RESULT OF
                                    ;IBM HOST CHAINED WRITE OPERATION
        DEFIN$    IS.BV,+05.        ;(A/D READ) AT LEAST ONE BAD VALUE
                                    ;WAS READ (REMAINDER MAY BE GOOD).
                                    ;BAD CHANNEL IS INDICATED BY A
                                    ;NEGATIVE VALUE IN THE BUFFER.
        DEFIN$    IS.DAO,+02.       ;SUCCESSFUL BUT WITH DATA OVERRUN
                                    ;(NOT TO BE CONFUSED WITH IE.DAO)
```

```
;
; TTY SUCCESS CODES
;
      DEFIN$   IS.CR,<15*400+1>   ;CARRIAGE RETURN WAS TERMINATOR
      DEFIN$   IS.ESC,<33*400+1>  ;ESCAPE (ALTMODE) WAS TERMINATOR
      DEFIN$   IS.CC,<3*400+1>    ;CONTROL-C WAS TERMINATOR
      DEFIN$   IS.ESQ,<233*400+1> ;ESCAPE SEQUENCE WAS TERMINATOR
      DEFIN$   IS.PES,<200*400+1> ;PARTIAL ESCAPE SEQUENCE WAS TERMINATOR
      DEFIN$   IS.EOT,<4*400+1>   ;EOT WAS TERMINATOR (BLOCK MODE INPUT)
      DEFIN$   IS.TAB,<11*400+1>  ;TAB WAS TERMINATOR (FORMS MODE INPUT)
      DEFIN$   IS.TMO,+2.         ;REQUEST TIMED OUT
      DEFIN$   IS.OOB,+3.         ;OUT OF BAND TERMINATOR (TERM IN HIGH BYTE)
      DEFIN$   IS.TMM,+4.         ;READ COMPLETED, MANAGEMENT MODE SEQ RCVD

;
; Professional Bisync Success Codes
;
      DEFIN$   IS.RVI,+2. ; DATA SUCC. XMITTED; HOST ACKED W/RVI
      DEFIN$   IS.CNV,+3. ; DATA SUCC. XMITTED; HOST ACKED W/CONVERSATION
      DEFIN$   IS.XPT,+5. ; DATA SUCC. RECVD IN TRANSPARENT MODE
;
; Professional Bisync Abort Codes
;
; These codes are returned in the high byte of the first word of the IOSB
; when the low byte contains IE.ABO.
;
      DEFIN$   SB.KIL,-1.  ; ABORTED BY IO.KIL
      DEFIN$   SB.ACK,-2.  ; ABORTED BECAUSE TOO MANY ACKS RECD OUT OF SEQ
      DEFIN$   SB.NAK,-3.  ; ABORTED BECAUSE NAK THRESHOLD EXCEEDED
      DEFIN$   SB.ENQ,-4.  ; ABORTED BECAUSE ENQ THRESHOLD EXCEEDED
      DEFIN$   SB.BOF,-5.  ; ABORTED BECAUSE OF IO.RLB BUFFER OVERFLOW
      DEFIN$   SB.TMO,-6.  ; ABORTED BECAUSE OF TIMEOUT
      DEFIN$   SB.DIS,-7.  ; ABORTED BECAUSE HOST DISCONNECTED W/ DLE, EOT

; ******
;
; THE NEXT AVAILABLE ERROR NUMBER IS: -105.
;
; *****
      .IF      EQ,$$MSG
      .MACRO   IOERR$ A
      .ENDM    IOERR$
      .ENDC
      .ENDM    IOERR$
```

```
;
; DEFINE THE DIRECTIVE ERROR CODES RETURNED IN THE DIRECTIVE STATUS WORD
;
; FILE CONTROL SERVICES (FCS) RETURNS THESE CODES IN THE BYTE F.ERR
; OF THE FILE DESCRIPTOR BLOCK (FDB).  TO DISTINGUISH THEM FROM THE
; OVERLAPPING CODES FROM HANDLER AND FILE PRIMITIVES, THE BYTE
; F.ERR+1 IN THE FDB WILL BE NEGATIVE FOR A DIRECTIVE ERROR CODE.
;
        .MACRO   DRERR$       $$$GBL
        .MCALL   .QIOE.,DEFIN$
        .IF      IDN,<$$$GBL>,<DEF$G>
        ...GBL=1
        .IFF
        ...GBL=0
        .ENDC
        .IIF     NDF,$$MSG,$$MSG=0

;
; STANDARD ERROR CODES RETURNED BY DIRECTIVES IN THE DIRECTIVE STATUS WORD
;
        .QIOE.   IE.UPN,-01.,<Insufficient dynamic storage> ; SEE ALSO IE.NDR
        .QIOE.   IE.INS,-02.,<Specified task not installed>
        .QIOE.   IE.PTS,-03.,<Partition too small for task>
        .QIOE.   IE.UNS,-04.,<Insufficient dynamic storage for send>
        .QIOE.   IE.ULN,-05.,<Un-assigned LUN>
        .QIOE.   IE.HWR,-06.,<Device handler not resident>
        .QIOE.   IE.ACT,-07.,<Task not active>
        .QIOE.   IE.ITS,-08.,<Directive inconsistent with task state>
        .QIOE.   IE.FIX,-09.,<Task already fixed/unfixed>
        .QIOE.   IE.CKP,-10.,<Issuing task not checkpointable>
        .QIOE.   IE.TCH,-11.,<Task is checkpointable>
        .QIOE.   IE.RBS,-15.,<Receive buffer is too small>
        .QIOE.   IE.PRI,-16.,<Privilege violation>
        .QIOE.   IE.RSU,-17.,<Resource in use>
        .QIOE.   IE.NSW,-18.,<No swap space available>
        .QIOE.   IE.ILV,-19.,<Illegal vector specified>
        .QIOE.   IE.ITN,-20.,<Invalid table number>
        .QIOE.   IE.LNF,-21.,<Logical name not found>
```

```
;
;
;
        .QIOE.    IE.AST,-80.,<Directive issued/not issued from AST>
        .QIOE.    IE.MAP,-81.,<Illegal mapping specified>
        .QIOE.    IE.IOP,-83.,<Window has I/O in progress>
        .QIOE.    IE.ALG,-84.,<Alignment error>
        .QIOE.    IE.WOV,-85.,<Address window allocation overflow>
        .QIOE.    IE.NVR,-86.,<Invalid region ID>
        .QIOE.    IE.NVW,-87.,<Invalid address window ID>
        .QIOE.    IE.ITP,-88.,<Invalid TI parameter>
        .QIOE.    IE.IBS,-89.,<Invalid send buffer size ( .GT. 255.)>
        .QIOE.    IE.LNL,-90.,<LUN locked in use>
        .QIOE.    IE.IUI,-91.,<Invalid UIC>
        .QIOE.    IE.IDU,-92.,<Invalid device or unit>
        .QIOE.    IE.ITI,-93.,<Invalid time parameters>
        .QIOE.    IE.PNS,-94.,<Partition/region not in system>
        .QIOE.    IE.IPR,-95.,<Invalid priority ( .GT. 250.)>
        .QIOE.    IE.ILU,-96.,<Invalid LUN>
        .QIOE.    IE.IEF,-97.,<Invalid event flag ( .GT. 64.)>
        .QIOE.    IE.ADP,-98.,<Part of DPB out of user's space>
        .QIOE.    IE.SDP,-99.,<DIC or DPB size invalid>
;
; SUCCESS CODES FROM DIRECTIVES - PLACED IN THE DIRECTIVE STATUS WORD
;
        DEFIN$    IS.CLR,0        ;EVENT FLAG WAS CLEAR
                                  ;FROM CLEAR EVENT FLAG DIRECTIVE
        DEFIN$    IS.SET,2        ;EVENT FLAG WAS SET
                                  ;FROM SET EVENT FLAG DIRECTIVE
        DEFIN$    IS.SPD,2        ;TASK WAS SUSPENDED
                                  ;
        DEFIN$    IS.SUP,3        ;LOGICAL NAME SUPERSEDED
                                  ;
        DEFIN$    IS.WAT,4        ;OPERATION INITIATED, WAIT FOR COMPLETION
                                  ;FROM "VAX-11 RSX" RMS-21 ELEP$ DIRECTIVE

;
;
        .IF       EQ,$$MSG
        .MACRO    DRERR$  A
        .ENDM     DRERR$
        .ENDC
        .ENDM     DRERR$
```

```
;
; DEFINE THE GENERAL I/O FUNCTION CODES - DEVICE INDEPENDENT
;
        .MACRO   FILIO$  $$$GBL
        .MCALL   .WORD.,DEFIN$
        .IF      IDN,<$$$GBL>,<DEF$G>
        .IFF
        ...GBL=0
        .ENDC
;
; GENERAL I/O QUALIFIER BYTE DEFINITIONS
;
        .WORD.   IQ.X,001,000      ;NO ERROR RECOVERY
        .WORD.   IQ.Q,002,000      ;QUEUE REQUEST IN EXPRESS QUEUE
        .WORD.   IQ.S,004,000      ;SYNONYM FOR IQ.UMD
        .WORD.   IQ.UMD,004,000    ;USER MODE DIAGNOSTIC STATUS REQUIRED
        .WORD.   IQ.LCK,200,000    ;MODIFY IMPLIED LOCK FUNCTION
;
; EXPRESS QUEUE COMMANDS
;
        .WORD.   IO.KIL,012,000    ;KILL CURRENT REQUEST
        .WORD.   IO.RDN,022,000    ;I/O RUNDOWN
        .WORD.   IO.UNL,042,000    ;UNLOAD I/O HANDLER TASK
        .WORD.   IO.LTK,050,000    ;LOAD A TASK IMAGE FILE
        .WORD.   IO.RTK,060,000    ;RECORD A TASK IMAGE FILE
        .WORD.   IO.SET,030,000    ;SET CHARACTERISTICS FUNCTION
;
; GENERAL DEVICE DRIVER CODES
;
        .WORD.   IO.WLB,000,001    ;WRITE LOGICAL BLOCK
        .WORD.   IO.RLB,000,002    ;READ LOGICAL BLOCK
        .WORD.   IO.LOV,010,002    ;LOAD OVERLAY (DISK DRIVER)
        .WORD.   IO.LDO,110,002    ;LOAD D-SPACE OVERLAY (DISK)
        .WORD.   IO.ATT,000,003    ;ATTACH A DEVICE TO A TASK
        .WORD.   IO.DET,000,004    ;DETACH A DEVICE FROM A TASK
;
; DIRECTORY PRIMITIVE CODES
;
        .WORD.   IO.FNA,000,011    ;FIND FILE NAME IN DIRECTORY
        .WORD.   IO.RNA,000,013    ;REMOVE FILE NAME FROM DIRECTORY
        .WORD.   IO.ENA,000,014    ;ENTER FILE NAME IN DIRECTORY
```

```
;
; FILE PRIMITIVE CODES
;
        .WORD.    IO.CLN,000,007    ;CLOSE OUT LUN
        .WORD.    IO.ULK,000,012    ;UNLOCK BLOCK
        .WORD.    IO.ACR,000,015    ;ACCESS FOR READ
        .WORD.    IO.ACW,000,016    ;ACCESS FOR WRITE
        .WORD.    IO.ACE,000,017    ;ACCESS FOR EXTEND
        .WORD.    IO.DAC,000,020    ;DE-ACCESS FILE
        .WORD.    IO.RVB,000,021    ;READ VIRITUAL BLOCK
        .WORD.    IO.WVB,000,022    ;WRITE VIRITUAL BLOCK
        .WORD.    IO.EXT,000,023    ;EXTEND FILE
        .WORD.    IO.CRE,000,024    ;CREATE FILE
        .WORD.    IO.DEL,000,025    ;DELETE FILE
        .WORD.    IO.RAT,000,026    ;READ FILE ATTRIBUTES
        .WORD.    IO.WAT,000,027    ;WRITE FILE ATTRIBUTES
        .WORD.    IO.APV,010,030    ;PRIVILEGED ACP CONTROL
        .WORD.    IO.APC,000,030    ;ACP CONTROL

;
;
        .MACRO    FILIO$  A
        .ENDM     FILIO$
        .ENDM     FILIO$
;
; DEFINE THE I/O FUNCTION CODES THAT ARE SPECIFIC TO INDIVIDUAL DEVICES
;
        .MACRO    SPCIO$  $$$GBL
        .MCALL    .WORD.,DEFIN$
        .IF       IDN,<$$$GBL>,<DEF$G>
        ...GBL=1
        .IFF
        ...GBL=0
        .ENDC

;
; I/O FUNCTION CODES FOR SPECIFIC DEVICE-DEPENDENT FUNCTIONS
;
        .WORD.    IO.WLV,100,001  ;(DECTAPE) WRITE LOGICAL REVERSE
        .WORD.    IO.WLS,010,001  ;(COMM.) WRITE PRECEDED BY SYNC TRAIN
        .WORD.    IO.WNS,020,001  ;(COMM.) WRITE, NO SYNC TRAIN
        .WORD.    IO.WAL,010,001  ;(TTY) WRITE PASSING ALL CHARACTERS
        .WORD.    IO.WMS,020,001  ;(TTY) WRITE SUPPRESSIBLE MESSAGE
        .WORD.    IO.CCO,040,001  ;(TTY) WRITE WITH CANCEL CONTROL-O
        .WORD.    IO.WBT,100,001  ;(TTY) WRITE WITH BREAKTHROUGH
        .WORD.    IO.WLT,010,001  ;(DISK) WRITE LAST TRACK
        .WORD.    IO.WLC,020,001  ;(DISK) WRITE LOGICAL W/ WRITECHECK
        .WORD.    IO.WPB,040,001  ;(DISK) WRITE PHYSICAL BLOCK
        .WORD.    IO.WDD,140,001  ;(FLOPPY DISK) WRITE PHYSICAL W/ DELETED DATA
        .WORD.    IO.RSN,140,002  ;(MSCP DISK) READ VOLUME SERIAL NUMBER
        .WORD.    IO.RLV,100,002  ;(MAGTAPE,DECTAPE) READ REVERSE
        .WORD.    IO.RST,001,002  ;(TTY) READ WITH SPECIAL TERMINATOR
        .WORD.    IO.RAL,010,002  ;(TTY) READ PASSING ALL CHARACTERS
        .WORD.    IO.RNE,020,002  ;(TTY) READ WITHOUT ECHO
        .WORD.    IO.RNC,040,002  ;(TTY) READ - NO LOWERCASE CONVERT
        .WORD.    IO.RTM,200,002  ;(TTY) READ WITH TIME-OUT
        .WORD.    IO.RDB,200,002  ;(CARD READER) READ BINARY MODE
        .WORD.    IO.SCF,200,002  ;(DISK) SHADOW COPY FUNCTION
        .WORD.    IO.RHD,010,002  ;(COMM.) READ, STRIP SYNC
```

```
.WORD.   IO.RNS,020,002  ;(COMM.) READ, DON'T STRIP SYNC
.WORD.   IO.CRC,040,002  ;(COMM.) READ, DON'T CLEAR CRC
.WORD.   IO.RPB,040,002  ;(DISK) READ PHYSICAL BLOCK
.WORD.   IO.RDF,240,002  ;(DISK) READ DISK FORMAT
.WORD.   IO.RLC,020,002  ;(DISK,MAGTAPE) READ LOGICAL W/ READCHECK
.WORD.   IO.ATA,010,003  ;(TTY) ATTACH WITH ASTS
.WORD.   IO.GTS,000,005  ;(TTY) GET TERMINAL SUPPORT CHARACTERISTICS
.WORD.   IO.R1C,000,005  ;(AFC,ADO1,UDC) READ SINGLE CHANNEL
.WORD.   IO.INL,000,005  ;(COMM.) INITIALIZATION FUNCTION
.WORD.   IO.TRM,010,005  ;(COMM.) TERMINATION FUNCTION
.WORD.   IO.RWD,000,005  ;(MAGTAPE,DECTAPE) REWIND
.WORD.   IO.SPB,020,005  ;(MAGTAPE) SPACE "N" BLOCKS
.WORD.   IO.RPL,020,005  ;(DISK) REPLACE LOGICAL BLOCK (RESECTOR)
.WORD.   IO.SPF,040,005  ;(MAGTAPE) SPACE "N" EOF MARKS
.WORD.   IO.STC,100,005  ;SET CHARACTERISTIC
.WORD.   IO.SMD,110,005  ;(FLOPPY DISK) SET MEDIA DENSITY
.WORD.   IO.SEC,120,005  ;SENSE CHARACTERISTIC
.WORD.   IO.RWU,140,005  ;(MAGTAPE,DECTAPE) REWIND AND UNLOAD
.WORD.   IO.SMO,160,005  ;(MAGTAPE) MOUNT & SET CHARACTERISTICS
.WORD.   IO.HNG,000,006  ;(TTY) HANGUP DIAL-UP LINE
.WORD.   IO.HLD,100,006  ;(TMS) HANGUP BUT LEAVE LINE ON HOLD
.WORD.   IO.BRK,200,006  ;(PRO/TTY) SEND SHORT OR LONG BREAK
.WORD.   IO.RBC,000,006  ;READ MULTICHANNELS (BUFFER DEFINES CHANNELS)
.WORD.   IO.MOD,000,006  ;(COMM.) SETMODE FUNCTION FAMILY
.WORD.   IO.HDX,010,006  ;(COMM.) SET UNIT HALF DUPLEX
.WORD.   IO.FDX,020,006  ;(COMM.) SET UNIT FULL DUPLEX
.WORD.   IO.SYN,040,006  ;(COMM.) SPECIFY SYNC CHARACTER
.WORD.   IO.EOF,000,006  ;(MAGTAPE) WRITE EOF
.WORD.   IO.ERS,020,006  ;(MAGTAPE) ERASE TAPE
.WORD.   IO.DSE,040,006  ;(MAGTAPE) DATA SECURITY ERASE
.WORD.   IO.RTC,000,007  ;READ CHANNEL - TIME BASED
.WORD.   IO.SAO,000,010  ;(UDC) SINGLE CHANNEL ANALOG OUTPUT
.WORD.   IO.SSO,000,011  ;(UDC) SINGLE SHOT, SINGLE POINT
.WORD.   IO.RPR,000,011  ;(TTY) READ WITH PROMPT
.WORD.   IO.MSO,000,012  ;(UDC) SINGLE SHOT, MULTI-POINT
.WORD.   IO.RTT,001,012  ;(TTY) READ WITH TERMINATOR TABLE
.WORD.   IO.SLO,000,013  ;(UDC) LATCHING, SINGLE POINT
.WORD.   IO.MLO,000,014  ;(UDC) LATCHING, MULTI-POINT
.WORD.   IO.LED,000,024  ;(LPS11) WRITE LED DISPLAY LIGHTS
.WORD.   IO.SDO,000,025  ;(LPS11) WRITE DIGITAL OUTPUT REGISTER
.WORD.   IO.SDI,000,026  ;(LPS11) READ DIGITAL INPUT REGISTER
.WORD.   IO.SCS,000,026  ;(UDC) CONTACT SENSE, SINGLE POINT
.WORD.   IO.REL,000,027  ;(LPS11) WRITE RELAY
.WORD.   IO.MCS,000,027  ;(UDC) CONTACT SENSE, MULTI-POINT
.WORD.   IO.ADS,000,030  ;(LPS11) SYNCHRONOUS A/D SAMPLING
.WORD.   IO.CCI,000,030  ;(UDC) CONTACT INT - CONNECT
.WORD.   IO.LOD,000,030  ;(LPA11) LOAD MICROCODE
.WORD.   IO.MDI,000,031  ;(LPS11) SYNCHRONOUS DIGITAL INPUT
.WORD.   IO.DCI,000,031  ;(UDC) CONTACT INT - DISCONNECT
.WORD.   IO.PAD,000,031  ;(PSI) DIRECT CONTROL OF X.29 PAD
.WORD.   HT.RPP,010,000  ;(PSI) RESET PAD PARAMETERS SUBFUNCTION
```

```
        .WORD.    IO.XMT,000,031  ;(COMM.) TRANSMIT SPECIFIED BLOCK WITH ACK
        .WORD.    IO.XNA,010,031  ;(COMM.) TRANSMIT WITHOUT ACK
        .WORD.    IO.INI,000,031  ;(LPA11) INITIALIZE
        .WORD.    IO.HIS,000,032  ;(LPS11) SYNCHRONOUS HISTOGRAM SAMPLING
        .WORD.    IO.RCI,000,032  ;(UDC) CONTACT INT - READ
        .WORD.    IO.RCV,000,032  ;(COMM.) RECEIVE DATA IN BUFFER SPECIFIED
        .WORD.    IO.CLK,000,032  ;(LPA11) START CLOCK
        .WORD.    IO.CSR,000,032  ;(BUS SWITCH) READ CSR REGISTER
        .WORD.    IO.MDO,000,033  ;(LPS11) SYNCHRONOUS DIGITAL OUTPUT
        .WORD.    IO.CTI,000,033  ;(UDC) TIMER - CONNECT
        .WORD.    IO.CON,000,033  ;(COMM.) CONNECT FUNCTION
                                  ;(VT11) - CONNECT TASK TO DISPLAY PROCESSOR
                                  ;(BUS SWITCH) CONNECT TO SPECIFIED BUS
                                  ;(COMM./PRO) DIAL TELEPHONE AND ORIGINATE
        .WORD.    IO.ORG,010,033  ;(COMM.) INITIATE CONNECTION IN ORIGINATE MODE
        .WORD.    IO.ANS,020,033  ;(COMM.) INITIATE CONNECTION IN ANSWER MODE
        .WORD.    IO.STA,000,033  ;(LPA11) START DATA TRANSFER
                                  ;(XJDRV) - SHOW STATE
        .WORD.    IO.DTI,000,034  ;(UDC) TIMER - DISCONNECT
        .WORD.    IO.DIS,000,034  ;(COMM.) DISCONNECT FUNCTION
                                  ;(VT11) - DISCONNECT TASK FROM DISPLAY PROCESSOR
                                  ;(BUS SWITCH) SWITCHED BUS DISCONNECT
        .WORD.    IO.MDA,000,034  ;(LPS11) SYNCHRONOUS D/A OUTPUT
        .WORD.    IO.DPT,010,034  ;(BUS SWITCH) DISCONNECT TO SPECIF PORT NO.
        .WORD.    IO.RTI,000,035  ;(UDC) TIMER - READ
        .WORD.    IO.CTL,000,035  ;(COMM.) NETWORK CONTROL FUNCTION
        .WORD.    IO.STP,000,035  ;(LPS11,LPA11) STOP IN PROGRESS FUNCTION
                                  ;(VT11) - STOP DISPLAY PROCESSOR
        .WORD.    IO.SWI,000,035  ;(BUS SWITCH) SWITCH BUSSES
        .WORD.    IO.CNT,000,036  ;(VT11) - CONTINUE DISPLAY PROCESSOR
                                  ;(XJDRV) - SHOW COUNTERS
        .WORD.    IO.ITI,000,036  ;(UDC) TIMER - INITIALIZE
;
; EXTENDED I/O FUNCTION
;
        .WORD.    IO.EIO,000,037  ;(TTY) TSA EXTENDED I/O
;
; PRO 300 SERIES BITMAP FUNCTIONS

;
;   NOTE: THESE FUNCTIONS ARE FOR DEC USE ONLY AND ARE SUBJECT TO CHANGE
;
        .WORD.    IO.RSD,030,014  ; READ SPECIAL DATA
        .WORD.    IO.WSD,010,013  ; WRITE SPECIAL DATA
        DEFIN$    SD.TXT,0        ; TEXT DATA TYPE FOR SPECIAL DATA
        DEFIN$    SD.GDS,1        ; GIDIS DATA TYPE FOR SPECIAL DATA

;
; PROFESSIONAL 300 BISYNC DRIVER (XJDRV) FUNCTIONS
;
        .WORD.    SB.PRT,020,003  ; ATTACH AS A PRINTER
        .WORD.    SB.CLR,010,036  ; CLEAR COUNTERS (IO.CNT SUBFUNCTION)
        .WORD.    SB.RDY,010,033  ; SET DEVICE STATE READY (IO.STA SUBFUNC)
        .WORD.    SB.NRD,020,033  ; SET DEVICE STATE NOT READY
        .WORD.    IO.LBK,000,035  ; PERFORM LOOPBACK TEST
        .WORD.    SB.CBL,010,035  ; PERFORM CABLE LOOPBACK TEST
        .WORD.    SB.CLK,020,035  ; DEVICE PERFORMS LINE CLOCKING
```

```
;
; COMMUNICATIONS FUNCTIONS
;
        .WORD.    IO.CPR,010,033  ;CONNECT NO TIME-OUTS
        .WORD.    IO.CAS,020,033  ;CONNECT WITH AST
        .WORD.    IO.CRJ,040,033  ;CONNECT REJECT
        .WORD.    IO.CBO,110,033  ;BOOT CONNECT
        .WORD.    IO.CTR,210,033  ;TRANSPARENT CONNECT
        .WORD.    IO.GNI,010,035  ;GET NODE INFORMATION
        .WORD.    IO.GLI,020,035  ;GET LINK INFORMATION
        .WORD.    IO.GLC,030,035  ;GET LINK INFO CLEAR COUNTERS
        .WORD.    IO.GRI,040,035  ;GET REMOTE NODE INFORMATION
        .WORD.    IO.GRC,050,035  ;GET REMOTE NODE ERROR COUNTS
        .WORD.    IO.GRN,060,035  ;GET REMOTE NODE NAME
        .WORD.    IO.CSM,070,035  ;CHANGE SOLO MODE
        .WORD.    IO.CIN,100,035  ;CHANGE CONNECTION INHIBIT
        .WORD.    IO.SPW,110,035  ;SPECIFY NETWORK PASSWORD
        .WORD.    IO.CPW,120,035  ;CHECK NETWORK PASSWORD
        .WORD.    IO.NLB,130,035  ;NSP LOOPBACK
        .WORD.    IO.DLB,140,035  ;DDCMP LOOPBACK

;
; ICS/ICR I/O FUNCTIONS
;
        .WORD.    IO.CTY,000,007  ;CONNECT TO TERMINAL INTERRUPTS
        .WORD.    IO.DTY,000,015  ;DISCONNECT FROM TERMINAL INTERRUPTS
        .WORD.    IO.LDI,000,016  ;LINK TO DIGITAL INTERRUPTS
        .WORD.    IO.UDI,010,023  ;UNLINK FROM DIGITAL INTERRUPTS
        .WORD.    IO.LTI,000,017  ;LINK TO COUNTER MODULE INTERRUPTS
        .WORD.    IO.UTI,020,023  ;UNLINK FROM COUNTER MODULE INTERRUPTS
        .WORD.    IO.LTY,000,020  ;LINK TO REMOTE TERMINAL INTERRUPTS
        .WORD.    IO.UTY,030,023  ;UNLINK FROM REMOTE TERMINAL INTERRUPTS
        .WORD.    IO.LKE,000,024  ;LINK TO ERROR INTERRUPTS
        .WORD.    IO.UER,040,023  ;UNLINK FROM ERROR INTERRUPTS
        .WORD.    IO.NLK,000,023  ;UNLINK FROM ALL INTERRUPTS
        .WORD.    IO.ONL,000,037  ;UNIT ONLINE
        .WORD.    IO.FLN,000,025  ;UNIT OFFLINE
        .WORD.    IO.RAD,000,021  ;READ ACTIVATING DATA

;
; IP11 I/O FUNCTIONS
;
        .WORD.    IO.MAO,010,007  ;MULTIPLE ANALOG OUTPUTS
        .WORD.    IO.LEI,010,017  ;LINK EVENT FLAGS TO INTERRUPT
        .WORD.    IO.RDD,010,020  ;READ DIGITAL DATA
        .WORD.    IO.RMT,020,020  ;READ MAPPING TABLE
        .WORD.    IO.LSI,000,022  ;LINK TO DSI INTERRUPTS
        .WORD.    IO.UEI,050,023  ;UNLINK EVENT FLAGS
        .WORD.    IO.USI,060,023  ;UNLINK FROM DSI INTERRUPTS
        .WORD.    IO.CSI,000,026  ;CONNECT TO DSI INTERRUPTS
        .WORD.    IO.DSI,000,027  ;DISCONNECT FROM DSI INTERRUPTS
        .WORD.    IO.RAM,000,032  ;READ ANALOG MAPPING TABLES
        .WORD.    IO.RLK,000,013  ;READ RESOURCE LINKAGES
        .WORD.    IO.EBT,000,011  ;CHECK EBIT STATUS

;
; PCL11 I/O FUNCTIONS
;
```

```
        .WORD.    IO.ATX,000,001  ;ATTEMPT TRANSMISSION
        .WORD.    IO.ATF,000,002  ;ACCEPT TRANSFER
        .WORD.    IO.CRX,000,031  ;CONNECT FOR RECEPTION
        .WORD.    IO.DRX,000,032  ;DISCONNECT FROM RECEPTION
        .WORD.    IO.RTF,000,033  ;REJECT TRANSFER

        .MACRO    SPCIO$  A
        .ENDM     SPCIO$
        .ENDM     SPCIO$
;
; DEFINE THE I/O CODES FOR USER-MODE DIAGNOSTICS.  ALL DIAGNOSTIC
; FUNCTIONS ARE IMPLEMENTED AS A SUBFUNCTION OF I/O CODE 10 (OCTAL).
;
        .MACRO    UMDIO$   $$$GBL
        .MCALL    .WORD.,DEFIN$
        .IF IDN   <$$$GBL>,<DEF$G>
...GBL=1
        .IFF
...GBL=0
        .ENDC

;
; DEFINE THE GENERAL USER-MODE I/O QUALIFIER BIT.
;
        .WORD.    IQ.UMD,004,000  ;USER-MODE DIAGNOSTIC REQUEST

;
; DEFINE USER-MODE DIAGNOSTIC FUNCTIONS.
;
        .WORD.    IO.HMS,000,010  ;(DISK) HOME SEEK OR RECALIBRATE
        .WORD.    IO.BLS,010,010  ;(DISK) BLOCK SEEK
        .WORD.    IO.OFF,020,010  ;(DISK) OFFSET POSITION
        .WORD.    IO.RDH,030,010  ;(DISK) READ DISK HEADER
        .WORD.    IO.WDH,040,010  ;(DISK) WRITE DISK HEADER
        .WORD.    IO.WCK,050,010  ;(DISK) WRITECHECK (NONTRANSFER)
        .WORD.    IO.RNF,060,010  ;(DECTAPE) READ BLOCK NUMBER FORWARD
        .WORD.    IO.RNR,070,010  ;(DECTAPE) READ BLOCK NUMBER REVERSE
        .WORD.    IO.LPC,100,010  ;(MAGTAPE) READ LONGITUDINAL PARITY CHAR
        .WORD.    IO.RTD,120,010  ;(DISK) READ TRACK DESCRIPTOR
        .WORD.    IO.WTD,130,010  ;(DISK) WRITE TRACK DESCRIPTOR
        .WORD.    IO.TDD,140,010  ;(DISK) WRITE TRACK DESCRIPTOR DISPLACED
        .WORD.    IO.DGN,150,010  ;DIAGNOSE MICRO PROCESSOR FIRMWARE
        .WORD.    IO.WPD,160,010  ;(DISK) WRITE PHYSICAL BLOCK
        .WORD.    IO.RPD,170,010  ;(DISK) READ PHYSICAL BLOCK
        .WORD.    IO.CER,200,010  ;(DISK) READ CE BLOCK
        .WORD.    IO.CEW,210,010  ;(DISK) WRITE CE BLOCK

;
; MACRO REDEFINITION TO NULL
;
        .MACRO    UMDIO$  A
        .ENDM
```

```
        .ENDM    UMDIO$
;
; HANDLER ERROR CODES RETURNED IN I/O STATUS BLOCK ARE DEFINED THROUGH THIS
; MACRO, WHICH THEN CONDITIONALLY INVOKES THE MESSAGE-GENERATING MACRO
; FOR THE QIOSYM.MSG FILE
;
        .MACRO   .IOER.  SYM,LO,MSG
        DEFIN$   SYM,LO
        .IF      GT,$$MSG
        .MCALL   .IOMG.
        .IOMG.   SYM,LO,<symbol>(MSG)
        .ENDC
        .ENDM    .IOER.

;
; I/O ERROR CODES ARE DEFINED THROUGH THIS MACRO, WHICH THEN INVOKES THE
; ERRQR MESSAGE-GENERATING MACRO; ERROR CODES -129 THROUGH -256
; ARE USED IN THE QIOSYM.MSG FILE
;
        .MACRO   .QIOE.       SYM,LO,MSG
        DEFIN$   SYM,LO
        .IF      GT,$$MSG)
        .MCALL   .IOMG.
        .IOMG.   SYM,<LO-128.>,<symbol>(MSG)
        .ENDC
        .ENDM    .QIOE.

;
; CONDITIONALLY GENERATE DATA FOR WRITING A MESSAGE FILE
;
        .MACRO   .IOMG.   SYM,LO,MSG
        .WORD    -^O<symbol>(LO)
        .ENABL   LC
        .ASCIZ   ^MSG^
        .DSABL   LC
        .EVEN
        .IIF     LT,^O<symbol>($$$MAX+<symbol>(LO)),$$$MAX=-^O<symbol>(LO)
        .ENDM    .IOMG.

;
; DEFINE THE SYMBOL SYM WHERE LO IS THE LOW-ORDER BYTE, HI IS THE HIGH BYTE
;
        .MACRO   .WORD.  SYM,LO,HI
        DEFIN$   SYM,<symbol>(HI*400+LO)
        .ENDM    .WORD.
        .DSABL   LC
```

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

RSX–11M–PLUS and Micro/RSX
I/O Drivers
Reference Manual
AA–JS11A–TC

READER'S COMMENTS

Your comments and suggestions are welcome and will help us in our continuous effort to improve the quality and usefulness of our documentation and software.

Remember, the system includes information that you read on your terminal: help files, error messages, prompts, and so on. Please let us know if you have comments about this information, too.

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

Did you find errors in this manual? If so, specify the error and the page number.

What kind of user are you? ___ Programmer ___ Nonprogrammer

Years of experience as a computer programmer/user: _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

--- Do Not Tear - Fold Here and Tape ---

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

--- Do Not Tear - Fold Here ---