Micro/RSX
I/O Drivers Reference Manual

Order No. AA-Z507B-TC

Micro/RSX Version 3.0

digital equipment corporation · maynard, massachusetts

First Printing, December 1983
Revised, July 1985

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Printed in U.S.A.

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

ZK2552

HOW TO ORDER ADDITIONAL DOCUMENTATION

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

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

In Canada call 613-234-7726 (Ottawa-Hull)
800-267-6146 (all other Canadian)

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire 03061

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

DIRECT MAIL ORDERS (CANADA)

Digital Equipment of Canada Ltd.
940 Belfast Road
Ottawa, Ontario K1G 4C2
Attn: A&SG Business Manager

DIRECT MAIL ORDERS (INTERNATIONAL)

Digital Equipment Corporation
A&SG Business Manager
c/o Digital's local subsidiary or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

CONTENTS

CONTENTS

PREFACE

MANUAL OBJECTIVES

The purpose of this manual is to provide all the information needed to use the I/O device drivers supplied as part of the Micro/RSX system.

INTENDED AUDIENCE

This manual is for experienced Micro/RSX programmers who want to take advantage of the time 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/M-PLUS and Micro/RSX Executive Reference Manual, have some experience using the Task Builder and either MACRO-11 or FORTRAN languages, and be familiar with the manuals describing their use.

STRUCTURE OF THE MANUAL

Chapter 1 provides an overview of Micro/RSX input/output operations. It introduces you to the use of logical unit numbers, directive parameter blocks, 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 through 7 describe the use of all device drivers supported by 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, if relevant
- The error and status conditions that the driver returns on acceptance or rejection of I/O requests
- Programming hints

Appendixes A through C provide quick reference material on I/O functions and status codes.

ASSOCIATED MANUALS

The following Micro/RSX manuals may be useful:

- RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual
- RSX-11M/M-PLUS and Micro/RSX Task Builder Manual
- PDP-11 MACRO-11 Language Reference Manual
- Micro/RSX Release Notes

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

CONVENTIONS USED IN THIS MANUAL

The following conventions are observed in this manual.

Convention	Meaning
[ ]	Square brackets; enclose optional syntax
{ }	Braces; indicate that one of the enclosed items must be selected.
...	Horizontal ellipsis; indicates that parameters have been omitted. In QIO macro calls in this manual, indicates that standard QIO parameters have been omitted.
,,,	Consecutive commas; used in coding examples to indicate null arguments. You may omit commas that indicate null trailing optional arguments.

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 through 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 per variable (that is, INTEGER*2) and all real arrays and variables are assumed to occupy two storage words per variable (that is, REAL*4).

SUMMARY OF TECHNICAL CHANGES

This revision of the Micro/RSX I/O Drivers Reference Manual reflects the following software technical changes and additions:

- A new I/O function, IO.EIO (Extended I/O), which contains new subfunctions, has been added to the full-duplex terminal driver.
- Support for the DZQ11 4-line terminal multiplexer, a DZV11 replacement, has been added to the full-duplex terminal driver.
- Support of the full-duplex terminal driver has been extended to allow its use as a Network Command Terminal.
- Support for the RC25 and RD52 disks, the KDA50 controller, and the RQDX2 controller (an MSCP RQDX1 replacement controller) has been added to the disk drivers.
- Support for the TK25 and TK50 magnetic tape drives has been added to the tape drivers.

In addition to these changes, Chapters 1 and 2 have been reorganized to make the information more easily accessible to the reader. Appendixes A and B have been updated to reflect the new I/O functions, subfunctions, and error codes that have been included.

CHAPTER 1

MICRO/RSX INPUT/OUTPUT

1.1 OVERVIEW OF MICRO/RSX I/O

This manual describes all of the Micro/RSX input/output (I/O) device drivers supported by the system. It gives driver characteristics, functions, error conditions, and provides programming hints for each. The drivers described in this manual are supplied by Digital Equipment Corporation as part of the system software to support associated devices. Devices not mentioned in this manual can be added to basic system configurations, but users must develop and maintain their own drivers for these devices. (See the RSX-11M-PLUS Guide to Writing an I/O Driver.)

The following loadable drivers are always loaded in the Micro/RSX system:

1. Terminal Driver (TTDRV.TSK)
2. Virtual Terminal Driver (VTDRV.TSK)
3. Winchester Fixed-Disk/ 5.25-inch Diskette Driver (DUDRV.TSK)
4. Reconfiguration Driver (RDDRV.TSK)
5. Null Device Driver (NLDRV.TSK)

The following loadable drivers are for the optional hardware devices and are not pre-loaded into the Micro/RSX system. These drivers are loaded automatically at system start-up, if the Autoconfigure task finds the associated device connected to the system.

1. Removable Cartridge Disk Driver (DLDRV.TSK)
2. TU58 Cartridge Tape Driver (DDDRV.TSK)
3. 8-Inch Diskette Driver (DYDRV.TSK)
4. Line Printer Driver (LPDRV.TSK)
5. TK25 Cartridge Tape Driver (MSDRV.TSK)
6. TK50 Cartridge Tape Driver (MUDRV.TSK)

Device drivers are operating system programs that communicate directly with the hardware devices connected to your microcomputer. A driver is the dividing line between Micro/RSX Executive system software and the hardware devices. Each driver recognizes specific types of hardware devices. This means each driver knows how to interpret the unique control signals sent by a particular hardware device and is programmed to notify the Executive when the device is ready to send or

receive data. At the operating system end, the Executive receives I/O requests from tasks and queues them as they are issued. These requests are then issued to the driver according to the relative priorities of the tasks that issued them. Tasks issue I/O requests in the form of macros called Queue I/O (QIO$) system macros.

QIO$s are used for MACRO-11 I/O programming at the driver level with all I/O requests sent directly to the Executive QIO$ processor. MACRO-11 I/O programming can also be done at a higher level by using either the File Control Services or Record Management Services. These services provide a high-level MACRO-11 I/O language user interface that generates the appropriate QIOs for a requested I/O operation.

All of the I/O services described in this manual are requested by the user in the form of QIO$ system macros. A function code included in the QIO$ macro indicates the particular input or output operation to be performed. I/O functions can be used to request such operations 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

Although input/output operations under Micro/RSX are extremely flexible and are as device- and function-independent as possible, there are device-specific input/output operations, such as disconnecting a terminal on a remote line and reading or writing a physical block, that can also be specified with QIO$ macros.

Device independence is achieved by associating logical units with physical units. Programs issue I/O requests to logical units that have been previously associated with particular physical device units. Each program or task is able to establish its own correspondence between physical device units and logical unit numbers (LUNs).

1.2 PHYSICAL, LOGICAL, AND VIRTUAL I/O

There are three possible modes in which an I/O transfer can take place: physical, logical, and virtual.

Physical I/O concerns reading and writing data in the actual physical units accepted by the hardware (for example, sectors on a disk). For most devices, physical I/O is identical to logical I/O in that a 256-word (512-byte) logical block exactly matches the size of a sector on disk. In the case of the RX02 flexible disk recorded in double density, the physical and logical blocks are different. Data for the RX02 is recorded in physical sectors of 128 words each. Therefore, logical blocks for the RX02 are made up of two physical sectors each.

Logical I/O concerns reading and writing data in blocks that are convenient for the operating system. In most cases, logical blocks map directly into physical blocks. For block-structured devices (for example, disks), logical blocks are numbered beginning at 0. For non-block-structured devices (for example, terminals), logical blocks are not addressable.

Virtual I/O concerns reading and writing data to open files. In this case, the Executive maps virtual blocks into logical blocks. For file-structured devices (disks or TU58 Tape), virtual blocks are the same size as logical blocks and are numbered starting from one (1); they 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 UNIT

This section describes the construction of the logical unit table and the use of logical unit numbers.

1.3.1 Logical Unit Number

A logical unit number (LUN), is a number associated with a physical device unit during I/O operations. For example, LUN 1 might be associated with one of the terminals in the system, LUNs 2, 3, 4, and 5 with TU58 tape 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 can change any LUN to physical device-unit association at almost any time. The flexibility of this association contributes heavily to system device independence.

A logical unit number is a short name that represents a logical-unit to physical-device-unit association. Once the association has been made, the LUN provides a direct and efficient mapping to the physical device unit, and eliminates the necessity to search the device tables whenever the system encounters a reference to a physical device unit.

You should remember that, although a LUN to physical device-unit association can be changed at any time, reassignment of a LUN at run time causes pending I/O requests for the previous LUN assignment to be canceled. It is your responsibility to verify that all outstanding I/O requests for a LUN have been serviced before that LUN is associated with another physical device unit.

1.3.2 Logical Unit Table

There is one Logical Unit Table (LUT) for each task running in an Micro/RSX operating system. 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 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 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 DCL command ASSIGN/REDIRECT. The second word of each 2-word entry is reserved for a pointer to the window block of the task that has a file open and mounted. 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 (by using the Unit and Device Control Blocks and other structures) to the LUN that the call specifies. The system does this by indexing into the LUT by 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 Logical Unit Table.

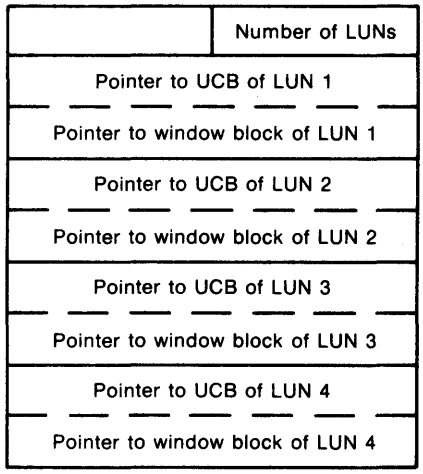


Figure 1-1 Logical Unit Table

1.3.3 Changing LUN Assignment

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 Task Builder. This option associates a physical device unit with a logical unit number referenced by the task being built.

- You or the system operator can issue a DCL DEASSIGN command followed by a DCL ASSIGN/REDIRECT command. 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 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 Micro/RSX system by submitting requests for I/O service as Queue I/O (QIO$) or Queue I/O And Wait (QIOW$) Executive macro. See the RSX-11M/M-PLUS and Micro/RSX Executive Reference Manual for a complete description of system macros.

The Micro/RSX operating system has 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 macro .MCALL in the source program. The macros reside in the System Macro Library (LB:[1,1]RSXMAC.SML). Section 1.6.4 describes the function of .MCALL.

In Micro/RSX, 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 it 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 a Micro/RSX system.

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 macro rejection by following the QIO$ macro with a MACRO-11 BCS instruction. (In a high-level language, the status is returned in a single-word variable which is specified as part of the call.)

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 Sections 1.4.3, 1.5.2, and 1.6.2) in the QIO$ request and should issue a Wait For Single Event Flag (WTSE$) Executive macro specifying 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 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
- Logical unit number 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 to which the Executive returns information indicating successful or unsuccessful completion
- Optional address of an asynchronous system trap 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 that you specify. If you want to specify only an I/O function code, a LUN, and an address for an asynchronous system trap service routine, issue the following:

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

where:

IO.ATT The I/O function code for attach.

6 The LUN.

,,,, Null arguments for the event flag number, the request priority, and the address of the I/O status block.

ASTOX The AST address.

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 comma if no parameters appear to the right of it. Therefore, you could issue the command above as follows, if you did not want the asynchronous system trap:

```
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. It appears as follows:

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

1.4.1.1 **Syntax Elements: Brackets [], Angle Brackets <>, Braces {} -** The following describes the syntax elements used in this manual to describe the QIO$ functions.

[ ] 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 <p1,...,p6 parameters. 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 1.6 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 form

```
IO.xxx
```

where:

xxx 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

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

where:

lun 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 this:

```
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 through 31. (decimal) 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 searches the library. You can obtain local symbolic definitions by the FILIO$ and SPCIO$ macros, which reside in the System Macro Library and 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 like this:

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

Because each task has its own 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 LUTs in table (if nonzero)/4
                          ;Each LUT is 2 words (4 bytes)
```

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 issued the QIOW$ macro, the Executive suspends task execution until the I/O completes. If the task issued 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$) macro (if group-global event flags are not being used), the Read Extended Flags (RDXF$) macro (for all event flags, including group-global event flags), or the Read Single Event Flag (RDEF$) macro.

If you specify an event flag number, it must be in the range 1 through 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 through 32 are local (specific to the issuing task); event flags 33 through 64 are global (shared by all tasks in the system). Event flags 65 through 96 are group-global event flags (shared by all tasks in the same user group). Flags 25 through 32 and 57 through 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 macro as if it were a simple QIO$ request.

1.4.1.5 **PRI Parameter** - The optional **pri** parameter is supplied only to make 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 I/O status block 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. It 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, specified by IOST) to determine whether a bad parameter has been detected:

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

The system object module library defines the correspondence between global symbolic names and I/O completion codes. The Task Builder 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 I/O status block. 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 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 occurs. If you want to interrupt your task to execute special code on completion of an I/O request, you can specify an asynchronous system trap 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, much like 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.

Sections 1.4.4 and 1.4.5 discuss asynchronous system traps, and the RSX-11M/M-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 Event

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 issues as a macro from within a task. This manual is concerned with the significant event caused by an I/O completion.

Significant events are normally set directly or indirectly 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 Flag

Event flags are a means by which tasks recognize specific events. (Tasks also use Asynchronous System Traps (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.

Ninety-six event flags are available to enable tasks to distinguish one event from another. Each event flag has a corresponding unique event flag number (efn). Numbers 1 through 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 through 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 through 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 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/M-PLUS and Micro/RSX Executive Reference Manual for a description of these directives.

The following 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 it 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 Trap

System traps can interrupt task execution and 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 traps differ as follows:

- Synchronous system traps (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/M-PLUS and Micro/RSX Executive Reference Manual.

- Asynchronous system traps (ASTs) signal the completion of an external event such as an I/O operation. An asynchronous system trap (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 Trap

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 macros can specify both an event flag and an AST; with these macros, 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 macros, 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 IOSB (specified by IOST) 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 (specified by IOST) 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 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) macros provide the mechanism for doing this.

Associating asynchronous system traps 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 provides for better overlap processing than the simple QIO$ and WTSE$ macros. Asynchronous system traps also provide an ideal mechanism for use in multiple buffering of I/O operations.

The Executive inserts all ASTs in a first-in-first-out 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 (PS), 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$S macro executed. The ASTX$S macro described in Section 1.6.7 of this manual issues the AST Service Exit macro. On AST service exit, control returns to another queued AST, to the executing task, or to another task waiting to run.

The RSX-11M/M-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 BLOCK

Directive Parameter Block (DPB) is a fixed-length area of contiguous memory that contains the arguments that you specify in a 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 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.

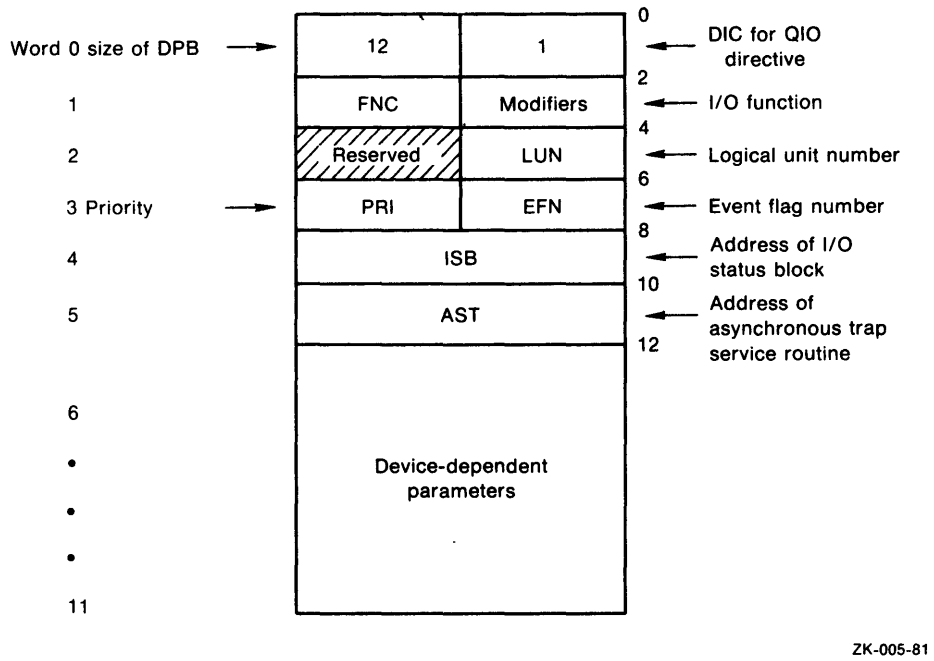


Figure 1-2 QIO$ Directive Parameter Block

1.5.1 I/O Packet

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 asynchronous system trap service routine.
- Return the I/O status.

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

1.6 I/O RELATED MACRO

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

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

- QIO$ (executed by using the DIR$ macro) generates a directive parameter block for the I/O request at assembly time, but does not provide the instructions necessary to execute the request. The QIO$ form is useful under the following conditions:
  - The task uses the DPB in several different places in the task.
  - The task modifies the DPB at run time.
  - The task references the DPB at run time.
- QIO$S generates a directive parameter block for the I/O request on the stack, and also generates code to execute the request. This is a useful form for reentrant, shareable code because QIO$S generates the DPB dynamically at execution time.
- QIO$C generates a directive parameter block for the I/O request at assembly time as well as generating code to execute the request. QIO$C generates the DPB in a separate program section called $DPB$$. QIO$C incurs little system overhead and it is useful when the task executes an I/O request from only one location. This manual uses the C form of the QIO$ macro in most of the examples in Chapter 1.

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 Assembler instructions such as MOV and MOVB. The following example references the same parameters in the three distinct forms of the macro call.

```
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/M-PLUS and Micro/RSX Executive Reference Manual describes the characteristics and use of these different forms.

The following section describes Executive directives issued as macros from within the task and Assembler directives:

- QIO$, which requests an I/O operation and supplies parameters for that request
- QIOW$, which is equivalent to QIO$ followed by WTSE$
- DIR$, which specifies the address of a directive parameter block as its argument, and generates code to execute the directive
- .MCALL, which makes all macros referenced during task assembly available from the System Macro Library

- ALUN$, which associates a logical unit number with a physical device unit at run time
- GLUN$, which requests that the information about a physical device unit to LUN association be returned to a buffer that you specify
- ASTX$S, which terminates execution of an asynchronous system trap (AST) service routine
- WTSE$, which instructs the system to block execution of the issuing task until a specified event flag is set

1.6.1 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 1.6.3) 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.

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

The QIOW$ macro is equivalent to a QIO$ followed by a WTSE$. It is more economical to issue a QIOW$ 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$. The reason efficiency increases is that system overhead is reduced.

The QIOW$ macro has the following syntax:

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

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

1.6.3 The DIR$ Macro: Executing a Directive

The DIR$ (execute directive) macro allows a task to reference a previously defined DPB. Issue it in the form:

```
DIR$   [addr],[err]
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
addr	The address of a directive parameter block 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	An optional argument which specifies the address of an error routine to which control branches if the macro is rejected. The branch occurs by means of a JSR PC, err if the C-bit is set, indicating rejection of the QIO$ macro.

In the following example, the DIR$ macro actually generates the code to execute the QIO$ directive. It provides no QIO$ parameters of its own, but references the QIO$ directive parameter block at address QIOREF by supplying this label as an argument.

```
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
```

1.6.4 The .MCALL Directive: Retrieving System Macros

.MCALL 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. .MCALL 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.

The following example illustrates the use of this directive:

```
         .MCALL  QIO$,QIO$S,DIR$,WTSE$S     ; MAKE MACROS AVAILABLE
             .
             .
             .
ATTACH:  QIO$S   #IO.ATT,#6,,, IOSB,#AST02  ; ATTACH DEVICE
             .
             .
             .
QIOREF:  QIO$    IO.RLB,6,,,IOSB,AST01,...  ; CREATE ONLY QIO$ DPB
             .
             .
             .
READ1:   DIR$    #QIOREF,DIRERR             ; ISSUE I/O REQUEST
             .
             .
             .
```

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.

1.6.5 The ALUN$ Macro: Assigning a LUN

The Assign LUN macro associates a logical unit number 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. Issue the macro from a MACRO-11 program in the following way:

```
ALUN$   lun,dev,unt
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number to be associated with the specified physical device unit. See Sections 1.3 and 1.4.1.3.
dev	The device name of the physical device or a logical device name assigned to a physical device (see the DCL ASSIGN command).
unt	The unit number of that device specified above.

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 disks, DECtape II, or terminals; it indicates the single available unit for devices without multiple units, such as card readers and line printers.

See the RSX-11M/M-PLUS Command Language Manual or the Micro/RSX User's Guide for a full description of the ASSIGN command.

The following example illustrates the use of the three forms of the ALUN$ macro.

```
;
; 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
```

1.6.5.1 **Physical Device Name** - The following list contains physical device names, listed alphabetically, that you may include as dev parameters:

Name	Device
DD	TU58 DECtape II
DL	RLV12/RL01/RL02 Cartridge Disk
DU	RC25 Disk Subsystem, RQDX1/RD51 Fixed-Media Disk, RD52 Fixed-Media Disk, RUX50 UNIBUS interface, and RX50 Flexible Disk
DY	RXV21/RX02 Flexible Disk
LP	LPV11/LP25/LP26 Line Printers and LN01/LN03 Laser Printer
MS	TSV05 Magnetic Tape, TK25 Cartridge Tape
MU	TK50 Cartridge Tape
NL	The Null Device
TT	Terminals (regardless of interface)
XE	RSX QIO DEUNA Driver
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)

1.6.5.2 **Nonphysical Device Name** - The following list contains names, listed alphabetically, that are not associated with a physical device but with a driver that interfaces with data structures instead of a real physical device (you may have heard these names referred to as pseudo devices but a pseudo device has a specific meaning in Micro/RSX - see Section 1.6.5.3):

Name	Device
HT	Network remote terminal
NL	The Null Device
NS	Network nonphysical device for NSP
NX	Network nonphysical device for DLX
RD	On-line reconfiguration pseudo-device
RT	Network Command Terminal
VT	Virtual terminal. Used by some offspring tasks as TI: for command and data I/O

1.6.5.3 **Pseudo-Device Name** - 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 one if a pseudo device name is used to reference a physical device.) Dynamic redirection of a physical device unit affects all tasks in the system; reassignment by means of the DCL DEASSIGN command affects only one task. The following pseudo-devices are supported, as indicated:

Code	Device
CL	Console listing, normally the line printer.
CO	Console output, normally the main operator's console.
LB	System library device, normally the device from which the system was bootstrapped. For example, LB: is the device that tasks such as TKB and MAC access for default library files.
SP	Spooling scratch disk device.
SY	User default device. SY: is normally the default login device.
TI	Pseudo-input terminal; TI0: is the terminal from which a task was requested. The pseudo-device TI cannot be redirected, since such redirection would have to be handled on a per-task rather than a system-wide basis (that is, change the TI device for one task without affecting the TI assignments for other tasks).

1.6.6 The GLUN$ Macro: Retrieving LUN Information

The Get LUN Information 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$ macro, the buffer contains the information listed in Table 1-1, as appropriate for the specific device. All three forms of the macro call may be used. It is issued from a MACRO-11 program in the following way:

```
GLUN$   lun,buf
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	The 6-word buffer to which information is returned.

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

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

Table 1-1
Get LUN Information

Numerical Offset			Symbolic Offset			Contents
Word	Byte	Bit	Word	Byte	Bit	
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 200 octal 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, 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, disk) (1 = yes)
		4			(DV.SDI)	Single directory device (for example, ANSI-standard magtape) (1 = yes)
		5			(DV.SQD)	Sequential device (for example, ANSI-standard magtape) (1 = yes)
		6			(DV.MSD)	Mass storage device (for example, disks and tapes) (1 = yes)
		7			(DV.UMD)	User-mode diagnostics supported (1 = yes)
		8			(DV.EXT)	22-bit direct addressing supported (1 = yes)

1. The following word and bit symbols shown in parentheses are symbols used in defining and referencing corresponding items in the device UCB.

(Continued on next page)

Table 1-1 (Cont.)
Get LUN Information

Numerical Offset			Symbolic Offset			Contents
Word	Byte	Bit	Word	Byte	Bit	
		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 items in the device UCB (1 = yes).
		14				(DV.Fll) Device mountable as a Files-11 device (for example disk) (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, and TU58 tape, this is the number of blocks (maximum logical block number plus one). For the proper use of the RXV21/RX02 flexible disk, it is important to be able to test G.LUCW+4 to determine the media density.

The example following illustrates the use of the three forms of the GLUN$ macro.

```
;
; 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
```

1.6.7 The ASTX$S Macro: Terminating AST Service

The ASTX$S macro terminates execution of an AST service routine. The Executive provides all forms of the macro. However, the S-form requires less space and executes at least as fast as the ASTX$ or ASTX$C form of the macro. Issue it as follows:

```
ASTX$S  [err]
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
err	An optional argument specifying the address of an error routine to which control branches if the macro 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 asynchronous system traps 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.)

1.6.8 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 macro call. Issue it as follows:

```
WTSE$   efn
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
efn	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. The following example illustrates task blocking until the specified event flag is set. This example also shows using three forms of the macro call.

```
;
        .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
        .
        .
        .
```

1.7 STANDARD I/O FUNCTION

You can specify a large number of input/output 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 and 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 NOP. 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 specified in the QIO$ macro call.

1.7.1 I/O Subfunction Bit

Most terminal QIO$ functions can be modified by a Logical OR of the symbolic name of a subfunction bit 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) that is specified in a Logical OR with the terminal-specific TF.ESQ subfunction (recognize escape sequences) would look like the following:

```
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. Often, you may want to use more than one subfunction bit. In this case, you may use several subfunction bits together in a Logical OR. As an example, for the terminal driver, 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 asynchronous system traps. To do this requires that you OR two subfunction bits with the IO.ATT function. If you do this, the QIO$ IO.ATT macro would look like the following:

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

If your task invokes a subfunction bit that is not supported on the system, the subfunction bit may be ignored or an error may occur.

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. The QIO$C IO.ATT macro can have the following format:

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

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 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 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, the Executive detaches the physical device unit.

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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 Micro/RSX QIO$ requests compatible with RSX-11D. A value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Section 1.4.1.6.
ast	For IO.ATT, 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. See the RSX-11M/M-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. Issue the QIO$C IO.DET macro as follows:

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

The LUN specifications of both IO.ATT and IO.DET must be the same, as in the following example, which also illustrates using S-forms of several macro calls.

```
        .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$S   #IO.RLB,#14.,... ; PRINT
        .
        .
        .
        QIO$S   #IO.DET,#14.    ; DETACH LINE PRINTER
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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 Micro/RSX QIO$ requests compatible with RSX-11D. A value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) 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.

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. 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 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.

For disk or TU58 tape I/O requests being processed when an IO.KIL is issued, the IO.KIL acts as a NOP. The request is allowed to complete. Because a disk operates quickly, IO.KIL causes the return of IS.SUC in the I/O status block.

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.

The QIO$C IO.KIL macro has the following format:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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 Micro/RSX QIO$ requests compatible with RSX-11D. A value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) 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.

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. The QIO$C IO.RLB macro has the following format:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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 Micro/RSX QIO$ requests compatible with RSX-11D. A value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) 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	The starting address of the data buffer. The address must be word aligned for certain drivers; stadd may be on a byte boundary.
size	The size of the stadd buffer in bytes. The buffer must be within the task's address space.
pn	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 them.

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 unless the task must issue subfunctions. However, if a task issues a virtual read for a file-structured device (disk or DECtape II), you must ensure that a file is open on the specified physical device unit. Issue IO.RVB as follows:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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 Micro/RSX QIO$ requests compatible with RSX-11D. A value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) 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	The starting address of the data buffer. The address must be word aligned for certain drivers; stadd may be on a byte boundary.
size	The size of the stadd buffer in bytes. The buffer must be within the task's address space.
pn	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.

If an IO.WVB write function is issued to a terminal, the Executive converts an IO.WVB to an IO.WLB request.

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

The QIO$C IO.WLB macro has the following format:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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 Micro/RSX QIO$ requests compatible with RSX-11D. A value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) 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	The starting address of the data buffer. The address must be word aligned for certain drivers; stadd may be on a byte boundary.
size	The size of the stadd buffer in bytes. The buffer must be within the task's address space.
pn	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.

NOTE

Any subfunction bits specified in the IO.WVB request (see Sections 2.3.1 and 3.3.1) are stripped off in this conversion.

All tasks should use IO.WVB rather than IO.WLB to 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.

The QIO$C IO.WVB macro has the following format:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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 Micro/RSX QIO$ requests compatible with RSX-11D. A value of 0 (or a null) should be used for this parameter.
isb	The address of the I/O status block (I/O status double-word) 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	The starting address of the data buffer. The address must be word aligned for certain drivers; stadd may be on a byte boundary.
size	The size of the stadd buffer in bytes. The buffer must be within the task's address space.
pn	One to four optional parameters, specifies 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. To perform a diagnostic function, you must specify in the QIO$ directive parameter block the logical OR of IQ.UMD and the function you want to perform. For example, to perform a diagnostic Read Logical Block operation, specify QIO$C IO.RLB!IQ.UMD as the I/O function code parameter to the QIO$ macro. You can execute standard I/O functions such as Read Logical Block, Write Logical Block, Attach to Device, and Detach from Device in diagnostic mode.

Support for user-mode diagnostics is always present for 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$.

To support user-mode diagnostics, the DV.UMD bit in the UCB must be set. DV.UMD is at offset U.CW1 in the UCB.

In addition to standard I/O functions, Micro/RSX provides the following device-dependent, user-mode diagnostic functions:

1. TU58 cartridge tape diagnostic functions
   - IO.BLS Block seek (explicit seek)
2. Magtape diagnostic functions
   - 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 the device for diagnostics using I/O function code IO.ATT!IQ.UMD. Execute the diagnostic functions and then detach.

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

- I/O buffer address
- I/O buffer size
- Double-precision logical block number
- User's register buffer address (the I/O driver copies its hardware registers to this buffer in the user's 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, QIOWAIT
                                    ;   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). No other error codes are returned. All error recovery is completely up to the user. Any errors that occur will not be 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$S  #14.,#"DL,#0                ; Associate DL0 with LUN 14
.
.
.
QIO$S   #IO.ATT!IQ.UMD,#14.         ; Attach DL for diagnostic I/O
.
.
.
QIO$S   #IO.WLB!IQ.UMD,#14.,,,,,<#$IOBUF,#512.,,#LBH,#LBL,#$RGBUF>
                                    ; Write logical block
.
QIO$S   #IO.WCK!IQ.UMD,#14.,,,,,<#$IOBUF,#512.,,#LBH,#LBL,#$RGBUF>
                                    ; Write check
.
.
QIO$S   #IO.DET!IQ.UMD,#14.         ; Detach DL
.
.
.
```

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 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 macro itself simply means that the macro was accepted and the I/O request was queued, not that the actual input/output 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. See Sections 1.4.5 and 1.6.7 for a description of AST service routines.

1.9.1 Return Codes

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

- Directive conditions, which indicate the acceptance or rejection of the QIO$ macro itself
- I/O status conditions, which 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 DIC number or DPB size
- Unassigned LUN
- Insufficient memory

The Executive returns a code indicating the acceptance or rejection of a directive to the Directive Status Word 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$ macro, identifying the address of a two-word I/O status block, 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 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 QIO$ Macro Conditions

Table 1-2 summarizes the macro conditions that your task may encounter by issuing QIO$ macros. The table lists acceptance condition first, followed by error codes indicating various reasons for rejection.

Table 1-2
Macro Conditions

Code	Reason
IS.SUC	**Directive accepted** The first six parameters of the QIO$ macro were valid, and sufficient dynamic memory was available to allocate an I/O packet.
IE.ADP	**Invalid address** The I/O status block or the QIO$ DPB was outside of the issuing task's address space or was not aligned on a word boundary.
IE.IEF	**Invalid event flag number** The efn specification in a QIO$ macro was less than 0 or greater than 96.
IE.ILU	**Invalid logical unit number** The lun specification in a QIO$ macro was invalid for the issuing task. For example, there were only 5 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 through 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$ macro was not associated with a physical device unit. Your task may recover from this error by issuing a valid Assign LUN (ALUN$) macro and then reissuing the rejected macro.
IE.UPN	**Insufficient dynamic memory** There was not enough dynamic memory to allocate an I/O packet for the I/O request. You can try again later by blocking the task with a Wait-For-Significant Event (WTSE$) macro. Note that WTSE$ is the only effective way for the issuing task to block its execution, because other macros for this purpose require dynamic memory for their execution (for example, Mark Time (MRKT$)).

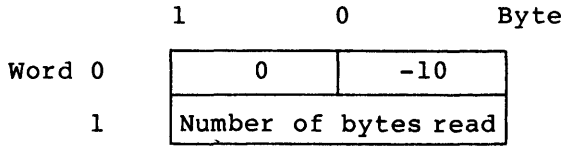
1.9.3 I/O Status Conditions

The contents of the 2-word I/O status block is explained next:

- The low-order byte of the first word receives a status code of the form IS.xxx or IE.xxx 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$ macro is omitted, this information is not returned.

The following illustrates an example 2-word I/O status block on completion of a terminal read operation:



where -10 is the status code for IE.EOF (end of file). If this code is returned, it indicates that input was terminated by typing 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 I/O status block with a symbolic value, as in the following:

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

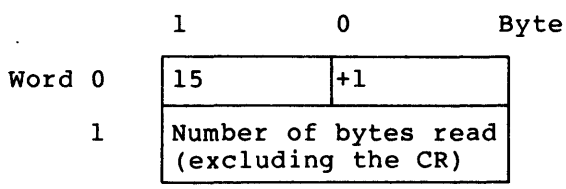
However, to test for certain types of successful completion of the I/O operation, the entire word value must be compared. For example, if a carriage return terminated a line of input from the terminal, a successful completion code of IS.CR is returned in the I/O status block. If an Escape (or Altmode) 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, Section B.1.2.)

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
```

In the case of a successful completion where the carriage return is the terminal indicator (IS.CR), the following illustrates the status block:



where 15 is the octal code for carriage return and +1 is the status code for successful completion.

Table 1-3 summarizes status codes that may be returned in the I/O status block specified in the QIO$ macro on completion of the I/O request. The codes described in Table 1-3 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 1-3 describes successful and pending codes first, then error codes.

Table 1-3
I/O Status Conditions

Code	Reason
IS.SUC	**Successful completion** The I/O operation specified in the QIO$ macro was completed successfully. The second word of the I/O status block 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$ macro has not yet been executed. The I/O status block is filled with 0s.
IE.ABO	**Operation aborted** The specified I/O operation was canceled with IO.KIL while in progress or while still in the I/O queue.
IE.ALN	**File already open** 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 parameter** 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 converter I/O operation.
IE.BBE	**Bad block on device** The disk sector (block) being read was marked as a bad block in the header word. Data cannot be written on or read from bad blocks.

(Continued on next page)

Table 1-3 (Cont.)
I/O Status Conditions

Code	Reason
IE.BLK	**Illegal block number** An invalid block number was specified for a file-structured physical device unit.
IE.BYT	**Byte-aligned buffer specified** Byte alignment was specified for a buffer, but only word (or double-word) 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.
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.
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$ macro was not ready to perform the desired I/O operation. This code is often returned as the result of an interrupt time-out; 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** Controller is physically unable to reach the location where input/output is to be performed on the device. The operation cannot be completed.

(Continued on next page)

Table 1-3 (Cont.)
I/O Status Conditions

Code	Reason
IE.IFC	**Illegal function** 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	**File not open** 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$ macro 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$ macro 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.
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.

(Continued on next page)

Table 1-3 (Cont.)
I/O Status Conditions

Code	Reason
	• You specified TF.XCC and AST2 in the same QIO$ request.
IE.VER	**Unrecoverable error** 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, 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 POWER-FAIL RECOVERY PROCEDURES FOR DISKS AND DECTAPE

Power-fail recovery recommendations for various devices are included in the following chapters. For disks, power recovery ASTs should be used. Prior to 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 I/O operations during power-fail 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.

1.11 MICRO/RSX DEVICES

The devices listed below are supported by Micro/RSX. Drivers are supplied for each of these devices, and I/O operations for them are described in detail in subsequent chapters of this manual.

1. A variety of terminals, including the following:
   - LA12 DECwriter
   - LA100 DECwriter
   - LA120 DECwriter III
   - LQP02 Letter-Quality Printer
   - LA50 Personal Printer

- VT100 Alphanumeric Display Terminal
- VT101 Alphanumeric Display Terminal
- VT102 Alphanumeric Display Terminal
- VT105 Alphanumeric Display Terminal
- VT125 Alphanumeric Display Terminal
- VT131 Alphanumeric Display Terminal
- VT132 Alphanumeric Display Terminal
- VT200 Series Alphanumeric Display Terminal

These terminals are supported on the following asynchronous line interfaces:

- DHV11 Asynchronous Communications Line Interface Multiplexer
- DZQ11 Asynchronous Communications Line Interface Multiplexer
- DLV11 Asynchronous Communications Line Interface Multiplexer
- DZV11 Asynchronous Communications Line Interface Multiplexer

2. The following disks:

- RLV12/RL01 or RL02 Cartridge Disk
- RQDX1/RQDX2 Controllers
- RD51 Fixed-Media Disk and RX50 Flexible Disk
- RD52 Fixed-Media Disk
- RXV21/RX02 Flexible Disk
- KDA50 Controller
- RQC25/RC25 Disk System
- RA60, RA80, and RA81 Disks

3. The following magnetic tapes:

- CPU Serial Line /TU58 DECtape II
- TSV05/TK25 Magnetic Tape Subsystem
- TQK50/TK50 Magnetic Tape Subsystem

4. The following line printers:

   - LPV11 Controller with LP25 or LP26 Line Printers
   - LN01/LN03 Laser Printers

5. The "Null Device," a software construct that facilitates eliminating unwanted output

6. Virtual Terminals

CHAPTER 2

FULL-DUPLEX TERMINAL DRIVER

2.1 INTRODUCTION

This chapter describes the full-duplex terminal driver (TTDRV.TSK) supplied with the Micro/RSX System. It describes the QIO$s that you can use to read from or write to the terminal in full-duplex mode. It also describes the subfunction bits that you can combine in a Logical OR with the QIO$ functions which modify and enhance the QIO$ functions.

The Micro/RSX full-duplex terminal driver has the following features:

- Full-duplex operation
- Type-ahead buffering
- Eight-bit characters
- Detection of hard receive errors
- Large byte transfer length (8128 bytes)
- Settable terminal characteristics
- Settable terminal types
- Optional time-out on solicited input
- Device-independent cursor control
- Redisplay of prompt buffer upon CTRL/R or CTRL/U
- Automatic XOFF character generation upon completion of a read (except when in the full-duplex mode), if requested
- Autobaud speed detection

2.1.1 The Full-Duplex Terminal Driver and Supported Devices

The full duplex terminal driver supports the following terminal devices:

- LA12 Portable Terminal
- LA100 DECprinter
- LA120 DECwriter
- LQP02 Letter-Quality Printer

- LA50 Personal Printer
- VT100 DECscope
- VT101 DECscope
- VT102 DECscope
- VT105 DECscope
- VT200 Series

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.

The terminal devices listed above are connected to your microcomputer through asynchronous serial line interfaces. The Micro/PDP-11 is delivered with two DLV11-like asynchronous single line interfaces, giving the capability for connecting two terminal devices, one of which is the console terminal. In addition to these two interfaces, there are asynchronous multiplexers that you can use for connecting multiple terminals to the Micro/RSX System: the DZQ11-CP multiplexer, the DZV11 multiplexer and additional DLV11 multiplexers or the DHV11 multiplexer. The DZQ11-CP Asynchronous Serial Line Multiplexer interfaces up to four asynchronous serial lines to the Q-Bus providing the capability of connecting four terminal devices to the system. It supports programmable baud rates. However, transmit and receive baud rates must be the same. The DHV11 interfaces eight asynchronous lines, providing the capability for connecting eight terminal devices; this multiplexer is for the full-duplex terminal driver only.

For additional information about the terminal devices and line interfaces see the MICRO/PDP-11 Handbook and the Terminals and Communications Handbook.

2.2 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 terminals. A setting of 1 indicates that the described characteristic is true for 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
5	0	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported

Bit	Setting	Meaning
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 buffer size (the width of the terminal carriage or display screen).

2.3 QIO$ MACRO

Standard QIO$ functions are general in use and may be used with any device, while device-specific QIO$ functions apply to specific devices or uses only.

2.3.1 Format of QIO$C for Standard Functions

The QIO$ macros for standard functions take the following forms:

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

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

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

2.3.2 Format of QIO$C for Device-Specific Functions

The QIO$ macro for device specific functions take the following forms:

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

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

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

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

```
QIO$C IO.HNG,...,

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

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 SF.SMC,...,<stadd,size>

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

Table 2-1 lists the standard and device-specific functions of the QIO$ macro that are valid for terminals. The standard functions are described in Chapter 1. Two device-specific functions, SF.SMC and SF.GMC, have nonstandard function names.

Table 2-1
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.

(continued on next page)

Table 2-1 (Cont.)
Standard QIO$ Functions for Terminals

Format	Function
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$C IO.WBT,...,<stadd,size,vfc>	WRITE logical block, break through any I/O conditions at terminal.
MISCELLANEOUS FUNCTIONS	
QIO$C IO.ATA,...,<ast,[parameter2] ,[ast2]>	ATTACH device, specify unsolicited-input-character 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$C SF.GMC,...,<stadd,size>	GET multiple characteristics.
QIO$C SF.SMC,...,<stadd,size>	SET multiple characteristics.
QIO$C IO.GTS,...,<stadd,size>	GET terminal support.
QIO$C IO.HNG,...	HANGUP remote line.

2.3.3 Parameters

The parameters for the various QIO$ macros have the following meanings:

Parameter	Meaning
ast	The entry point for an unsolicited input-character AST.
parameter2	A number that you can specify in your task to identify this terminal device as the input source upon entry to an unsolicited character AST routine.
ast2	The entry point for a CTRL/C AST.
pradd	The starting address of the byte buffer where the prompt is stored.
prsize	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.
stadd	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.
size	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.
table	The address of the 16-word user-defined terminator table.
tmo	An optional time-out count specified in 10-second intervals. (For IO.EIO, the interval is specified in seconds.) Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) 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(decimal) intervals (or 255(decimal) 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	The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use. The vfc parameter specifies cursor position. The parameter is interpreted 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).

Parameter	Meaning
	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. Table 2-9 is a summary of all 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.

2.3.4 Subfunction Bits

Most of the device-specific functions supported by terminal drivers described in this section are selected by using subfunction bits. One or more functions can be achieved by the Logical OR of the subfunction with the function, as IO.RPR!TF.RNE (read logical block after prompt with no echo). Specifying the function and subfunction in this way combines as an OR their relative bits in a QIO$ function. Table 2-2 contains a listing of QIO$ functions and relative subfunction bits that can be issued.

Each subfunction bit and subfunction selected when it is included in a QIO$ function is listed with its symbolic name and meaning as follows:

Subfunction	Meaning
TF.AST	Unsolicited-Input-Character AST - For IO.ATT or IO.ATT!TF.ESC, ast specifies the address of an 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.
TF.CCO	Cancel CTRL/O - Allows writing 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 from the terminal are returned to the task. Otherwise, ESC is a line terminator. This subfunction is for use with IO.ATA or IO.ATT.
TF.NOT	Notification Of Unsolicited Input - Unsolicited input causes an AST and entry into the AST service routine in the task. When the 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).

Subfunction	Meaning
	Using the TF.NOT subfunction allows a task to monitor more than one terminal for unsolicited input without the need to continuously read each terminal 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.RAL	Read All Characters (Pass All) - 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 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. When included in this manner, TF.RCU causes the driver to first save the current cursor position, then position the cursor and output the specified buffer, and, finally, 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 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 upper case; it is for use with the extended I/O (IO.EIO) function only.
TF.RNE	Read With No Echo - 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 - Read and pass through CTRL/U, CTRL/R, and DELETE characters as normal characters. This subfunction is for use with the extended I/O (IO.EIO) function only.

Subfunction	Meaning

TF.RPR Read After Prompt (For Extended I/O (IO.EIO) 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 functions 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 an 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 - Passes all characters except XON/XOFF. Allows the passage of all characters to the requesting task. The characteristic TC.8BC, when set, allows the driver to pass eight 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 not interpreted by the driver. This subfunction is for use with the extended I/O (IO.EIO) function only.

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

Subfunction	Meaning
TF.RTT	Read With The Terminator Table That You Specify - Use this subfunction with the IO.EIO extended I/O function only. Control characters function normally with this 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 subfunctions TF.RNF or TF.RAL, or the characteristic TC.PTH. The terminator table (a bit mask table) length can be from 1 through 32(decimal) 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 through 255(decimal), the characteristic TC.8BC must be set.
TF.TMO	Read With Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. (For IO.EIO, the interval is specified in seconds.) Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) 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(decimal) intervals (or 255(decimal) 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 - Allows reading terminator characters from the terminal without their being echoed on the terminal screen as they are entered. Use this subfunction with the extended I/O function IO.EIO.
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 input/output wrap around has been selected.
TF.WBT	Break-through 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 break-through write is the next write issued. Therefore, the TF.WBT subfunction cannot break

Subfunction	Meaning
	through another break-through write that is in progress. The effect of this is that a CTRL/S can stop break-through write functions. Thus, it may be desirable for tasks to time out on break-through operations. If a read is currently posted, the break-through write proceeds, and an automatic CTRL/R is performed to redisplay any input that was received before the break-through 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. Break-through write may be issued by a privileged task only. (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.
TF.XCC	Exclude CTRL/C or Abort Active Task -Use the TF.XCC subfunction 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, 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 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.

Table 2-2 lists subfunction bits that can be executed in a Logical OR with QIO functions.

The following example is a QIO request using more than one subfunction bit: a nonechoed read (TF.RNE), terminated by a special terminator character (TF.RST), and 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.

2.4.1 Functions and Allowed Subfunctions

Any given function except SF.GMC, SF.SMC, IO.EIO, and IO.GTS can be issued by using a Logical OR of a particular subfunction bit with another QIO function. Table 2-2 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 2-2
Subfunction Bits - Summary

Function	Equivalent Subfunctions	Allowed Subfunctions
STANDARD FUNCTIONS		
IO.ATT		TF.AST, TF.ESQ
IO.DET		
IO.KIL		
IO.RLB		TF.RAL, TF.RNE, TF.RST, TF.TMO, TF.XOF
IO.RVB[1]		TF.RAL, TF.RNE, TF.RST, TF.TMO, TF.XOF
IO.WLB		TF.CCO, TF.RCU, TF.WBT, TF.WAL
IO.WVB[1]		TF.CCO, TF.RCU, TF.WAL, TF.WBT
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

1. Subfunctions are stripped off if they are specified with IO.RVB or IO.WVB.

2. TF.RLB or TF.WLB but not both must be used with IO.EIO.

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

2.4.2 QIO$C IO.ATA - Attach a Terminal with ASTs

The QIO$ IO.ATA macro attaches the terminal and identifies ast and ast2 as entry points for unsolicited input-character ASTs that processes these characters. With ast and ast2, IO.ATA specifies asynchronous system traps (ASTs) to process unsolicited input characters (other than CTRL/Q, CTRL/S, CTRL/X, or CTRL/O). A minimum of one AST parameter (ast or ast2) is required.

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

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

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

The format of the QIO$C IO.ATA macro is as follows:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	The entry point for an unsolicited input-character AST. Either ast 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 has been specified, after the AST has been effected, the AST becomes "disarmed" until a read request is issued by the task. If multiple unsolicited input characters are received before the read request is issued, they are stored in the type-ahead buffer of attached terminals until they are read by the task. Once the read

Parameter	Meaning

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. The terminal driver discards all unsolicited characters from an unattached, slaved terminal. If TF.NOT is not specified, every usolicited 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 Parameter2 is located in the high byte of SP+00. It is a value that you can specify to identify individual terminals in a multiterminal environment.

ast2 The entry point for an unsolicited CTRL/C AST.

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

SP+10	Event flag mask word
SP+06	PS of task prior to AST
SP+04	PC of task prior to AST
SP+02	Task's directive status word
SP+00	Unsolicited character in low byte; parameter2, in the high byte, is a user-specified value that can be used to identify individual terminals in a multiterminal environment

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.

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 continuously read each terminal for possible unsolicited input. Note the the TF.NOT subfunction cannot be used with the CTRL/C AST; an unsolicited CTRL/C character flushes the type-ahead buffer.

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

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
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 for solicited input only (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 or 176, octal) 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 ESC sent by the terminal acts as a line terminator. The QIO$C SF.SMC function or the DCL SET /ESCAPE command declare the terminal capable of generating escape sequences (see Table 2-4 in Section 2.4.14, 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 continuously read each terminal 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 - You can use the TF.XCC subfunction 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, 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.

2.4.3 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.

The format of the QIO$ IO.CCO macro is as follows:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	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.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	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	The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use. The vfc parameter specifies cursor position. The parameter is interpreted 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

Parameter	Meaning

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.

Table 2-9 is a summary of all 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.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning

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 input/output wraparound has been selected.

TF.WBT Write Breakthrough - 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 write breakthrough is the next write issued. Therefore, the TF.WBT subfunction cannot break through another write breakthrough that is in progress. The effect of this is that a CTRL/S can stop write breakthrough 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.

Break-through write may be issued by a privileged task only. (The DCL command BRO (broadcast) uses IO.WBT.)

2.4.4 QIO$C IO.EIO - Extended I/O Functions

The IO.EIO function allows the use of additional I/O subfunctions. The QIO$ design 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 new I/O subfunction bits. Figure 2-1 shows the structure of the Item List 1 buffer required for the subfunction TF.RLB. Also, Figure 2-2 shows the structure of the Item List 2 buffer required for the subfunction TF.WLB.

The QIO$C IO.EIO 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 used in a Logical OR together.

NOTE

The IO.EIO function will not work if your terminal has been set as a remote terminal (RT:) to another system. That is, after entering

```
>SET HOST xxxxx
```

and logging into an RT:, the terminal driver will reject a QIO issuing an extended I/O request from the RT:.

The QIO$C IO.EIO macro has either one of the following formats:

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

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

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:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	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.
stadd	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	The size of the item list in bytes. The specified size for the IO.EIO!TF.RLB function must be 24 decimal bytes. The specified size for the IO.EIO!TF.WLB function must be 10 decimal bytes. The item list must be within the task's address space.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
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.
TF.CCO	Cancel CTRL/O - The driver writes a logical block of data to the terminal 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.
TF.RAL	Read All Characters (Pass All) - 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 not interpreted by the driver.

Subfunction	Meaning

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. When included in this manner, TF.RCU causes the driver to first save the current cursor position, then position the cursor and output the specified buffer, and, finally, 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 at the start of the read as an input line on the terminal. You may change this line or use it as input to the system. This subfunction is for 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 and 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 upper case. This subfunction is used with the extended I/O (IO.EIO) function only.

TF.RNE Read With No Echo - 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 - Read and pass through CTRL/U, CTRL/R, and DELETE characters as normal characters. This subfunction is for use with the extended I/O (IO.EIO) function only.

TF.RPR Read After Prompt (for 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 functions 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 an 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 - Passes all characters except XON/XOFF. The characteristic TC.8BC, when set, allows the driver to pass eight 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 not interpreted by the driver. This subfunction modifier is for use with the IO.EIO!IO.RLB function only.

TF.RST Read With Special Terminators - Certain special characters terminate the read. These characters are in the ranges 0-037 and 175-177. The driver does not interpret the terminating character. For example, a DELETE or RUBOUT (177) does not erase, and a CTRL/C does not produce a CLI prompt. Also CTRL/U and CTRL/R do not perform their usual functions. 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 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.

TF.RTT Read With Specified Terminator Table - Use this subfunction with the IO.EIO extended I/O function only. Control characters function normally with this 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 through 32(decimal) 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 through 255(decimal), the charateristic TC.8BC must be set.

TF.TMO Read With Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time.

Specify the time-out count in seconds. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) intervals.

Subfunction	Meaning
TF.TMO (Cont.)	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(decimal) 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 - Allows reading terminator characters from the terminal without their being echoed on the terminal screen as they are entered. Use this subfunction with the extended I/O function IO.EIO.
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 input/output wraparound has been selected.
TF.WBT	Write Breakthrough - 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 write breakthrough is the next write issued. Therefore, the TF.WBT subfunction cannot break through another write breakthrough that is in progress. The effect of this is that a CTRL/S can stop write breakthrough 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. Break-through write may be issued by a privileged task only.
TF.WIR	Write With Input Redisplayed - 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 - 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.

2.4.4.1 Item List 1 for IO.EIO!TF.RLB - The structure of the Item List 1 Buffer is shown in Figure 2-1. Modifier word 2 is currently not used but must be zero. All the other fields in the item list must be present, but need not contain any specific information except that pertinent to the function being performed. Thus, if a read with prompt (TF.RPR) is not being performed, words 10, 12 and 14 are not used. Item List 1 has the following form:

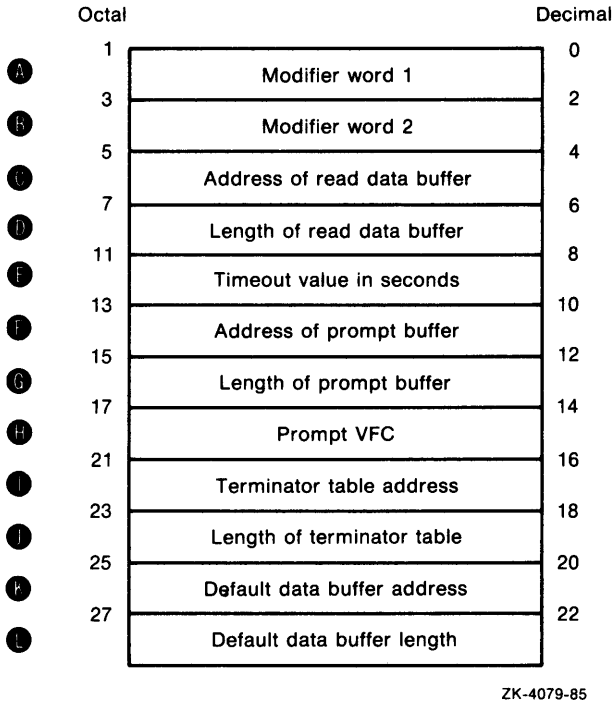


Figure 2-1 Structure of the Item List 1 Buffer

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

B Currently must be zero.

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

D 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.

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

F This field contains the starting address of the prompt buffer. The prompt buffer may be on a byte boundary. For use with TF.RPR, which must be in modifier word 1.

G 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.

H For use with TF.RPR. The vfc parameter specifies cursor position. The parameter is interpreted 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. 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.

Table 2-9 is a summary of all 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.

I For use with TF.RTT. TF.RTT must be in modifier word 1. The table (1 to 32(decimal) bytes) starts at the address specified by the table address. The first word contains bits that represent the first 16 ASCII character codes (0-17); similarly, the second word contains bits that represent the next 16 character codes (20-37), and so forth, through the sixteenth word, bit 15, which represents character code 377. For example, to specify the % 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 40-57.

The terminal must be set to read-pass-all operation (TC.BIN=1), or 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

J Length of the terminator table specified in I.

K 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.

L For use with TF.RDI. This word contains the length of the buffer at the address specified in K.

2.4.4.2 **Item List 2 for IO.EIO!TF.WLB** - The structure of the Item List 2 Buffer is shown in Figure 2-2. Modifier word 2 is currently not used but must be 0. All the other fields in the item list must be present. Item list 2 is shown in Figure 2-2.

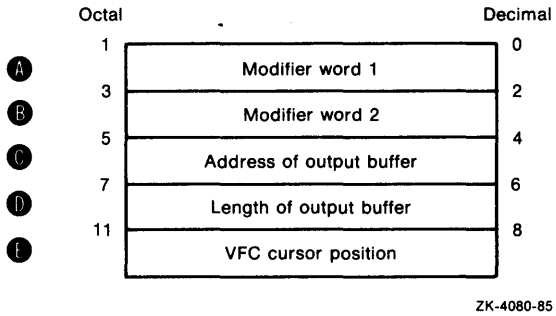


Figure 2-2 Structure of the Item List 2 Buffer

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

(B) Currently must be zero.

(C) The starting address of the write data buffer. The address may be on a byte boundary.

(D) The size of the stadd 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.

(E) The vfc parameter specifies cursor position. The parameter is interpreted 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. 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.

Table 2-9 is a summary of all 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.

2.4.5 QIO$C IO.GTS - Get Terminal Support

This function is a get terminal support request that returns information to a 4-word buffer specifying which features are part of the terminal driver. Only two of these words are currently defined. Table 2-3 gives details for these words.

The format of the QIO$C IO.GTS macro is as follows:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	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.
stadd	The starting address of the data buffer. The address must be word aligned.
size	The size of the stadd data buffer in bytes. The specified size must be four bytes. The buffer must be within the task's address space. The size must be an even value.

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 2-3); TC.xxx (Table 2-4); T.xxxx (Table 2-5); and the SE.xxx status returns described in Table 2-6, Section 2.5. These symbols may be defined locally within a code module by using:

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

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

Octal values shown for the symbols are subject to change. Therefore, it is recommended that only the symbolic names be used.

Table 2-3
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
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 11D/IAS IO.KIL
4	20	F2.ALT	ALTmode is echoed
5	40	F2.SFF	Formfeed 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

2.4.6 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. This function has no parameters.

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.

The format of the QIO$C IO.HNG macro is as follows:

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

Parameters:

The parameters have the following meaning:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	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.

2.4.7 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 in a Logical OR with the subfunction bit TF.RAL. The IO.RAL function can be terminated by a full character count only (input buffer full).

The format of QIO$C IO.RAL is as follows:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	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.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	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	The optional time-out count for use with the TF.TMO subfunction.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.RNE	Read With No Echo - 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.RST	Read With Special Terminators - Certain special characters terminate the read. These characters are in the ranges 0-037 and 175-177. The driver does not interpret the terminating character. For example, a DELETE or RUBOUT (177) does not erase, and a CTRL/C does not produce a CLI prompt, or, if CTRL/C abort is enabled, abort tasks active at the terminal. Also CTRL/U and CTRL/R do not perform their usual functions. 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 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 Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) 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(decimal) 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.

2.4.8 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 immediate display. You can use this feature when typing sensitive information (for example, a password or combination).

(Note that the no-echo mode can also be selected with the SF.SMC function; see Table 2-4, 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.

The format of the QIO$C IO.RNE macro is as follows:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	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.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	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	The optional time-out count for use with the TF.TMO subfunction.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
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 eight bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and 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 - Certain special characters terminate the read. These characters are in the ranges 0-037 and 175-177. The driver does not interpret the terminating character. For example, a DELETE or RUBOUT (177) does not erase, and a CTRL/C does not produce a CLI prompt, or, if CTRL/C abort is enabled, abort tasks active at the terminal. Also CTRL/U and CTRL/R do not perform their usual functions. 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 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 Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) 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(decimal) 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.RPR - Send Prompt, Then Issue Read

The QIO$C IO.RPR (Read After Prompt) 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 (to 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 using 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.WAL/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 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 used in 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 read subfunction bits TF.RAL, TF.RNE, TF.TMO, 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.

The format of the QIO$C IO.RPR macro is as follows:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.

Parameter	Meaning
ast	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.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	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	The optional time-out count for use with the TF.TMO subfunction.
pradd	The starting address of the byte buffer where the prompt is stored.
prsize	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	The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use. The vfc parameter specifies cursor position. The parameter is interpreted 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. 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. Table 2-9 is a summary of all 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.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.BIN	Binary Prompt (send prompt as pass all) - As used in IO.RPR, 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 eight bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and 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- -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.RST	Read With Special Terminators - Certain special characters terminate the read. These characters are in the ranges 0-037 and 175-177. The driver does not interpret the terminating character. For example, a DELETE or RUBOUT (177) does not erase, and a CTRL/C does not produce a CLI prompt, or, if CTRL/C is enabled, abort tasks active at the terminal. Also CTRL/U and CTRL/R do not perform their usual functions. 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 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 Real All and Read With Special Terminators together. Obscure problems can result if you use them in this way.

Subfunction	Meaning
TF.TMO	Read With Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) 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(decimal) 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.

2.4.10 QIO$C IO.RST - Read Logical Block with Special Terminators

Issue IO.RST to read a block of data from the specified physical device unit. This function is equivalent to an IO.RLB!TF.RST. Certain special characters in the ranges 0-037 and 175-177 terminate the read. The driver does not interpret the terminating character. For example, a DELETE or RUBOUT (177) does not erase, and a CTRL/C does not produce a CLI prompt, or, if CTRL/C is enabled, abort tasks active at the terminal. Also CTRL/U and CTRL/R do not perform their usual functions. 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 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 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.

The format of QIO$C IO.RST is as follows:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	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.

Parameter	Meaning
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	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	The optional time-out count for use with the TF.TMO subfunction.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
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 eight bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and 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 - 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.TMO	Read With Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) 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(decimal) 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.

2.4.11 QIO$ IO.RTT - Read with Terminator Table

The IO.RTT function reads characters in a manner like the IO.RLB function, except that a character that you have specified previously terminates the read operation. The specified character's code can range from 0 through 377 (octal). You can specify it by setting a bit in a 16-word table 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 (0-17); similarly, the second word contains bits that represent the next 16 character codes (20-37), and so forth, through the sixteenth word, bit 15, which represents character code 377. For example, to specify the % 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 40-57.

If you use the IO.EIO!TF.RLB function, the modifier TF.RTT allows you to specify the length of the table from 1 to 32(decimal) bytes.

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 eight bits (TC.8BC), as listed in Table 2-4.

The optional timeout count parameter may be included as desired.

The format of QIO$C IO.RTT is as follows:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	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.

Parameter	Meaning
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	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	The optional time-out count for use with the TF.TMO subfunction.
table	The address of the 16-word special terminator table that you create in your task.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
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 eight bits. For example, CTRL/C, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z are passed to the task and 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. When included in this manner, TF.RCU causes the driver to first save the current cursor position, then position the cursor and output the specified buffer, and, finally, restore the cursor to the original (saved) position once the output transfer has been completed.
TF.RNE	Read With No Echo - 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.TMO	Read With Time-Out - This subfunction allows the use of the tmo parameter to require input from the terminal within a specified time. Specify the time-out count in 10-second intervals. Time-out is the maximum time allowed between two input characters before the read is aborted. The maximum time-out value is 255(decimal) 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(decimal) 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.

2.4.12 QIO$C IO.WAL - Write a Logical Block and Pass all Bits

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 input/output wraparound has been selected.

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

The format of the QIO$C IO.WAL macro is as follows:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	The logical unit number of the associated physic device unit to be accessed by the I/O request. For more information refer to Chapter 1.
efn	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	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.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.
size	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.

Parameter	Meaning
vfc	The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use. The vfc parameter specifies cursor position. The parameter is interpreted 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. 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. Table 2-9 is a summary of all 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.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
TF.CCO	Cancel CTRL/O - 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, 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. When included in this manner, TF.RCU causes the driver to first save the current cursor position, then position the cursor and output the specified buffer, and, finally, 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.

Subfunction	Meaning
TF.WBT	Write Breakthrough - 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 write breakthrough is the next write issued. Therefore, the TF.WBT subfunction cannot break through another write breakthrough that is in progress. The effect of this is that a CTRL/S can stop write breakthrough functions. Thus, it may be desirable for tasks to time out on breakthrough write operations. If a read is currently posted, the write breakthrough 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. Break-through write may be issued by a privileged task only. (The DCL command BRO (broadcast) uses IO.WBT.) 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.

2.4.13 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.

- 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 rubbed out.

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

Breakthrough write may be issued by a privileged task only.

The format of the QIO$C IO.WBT macro is as follows:

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

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1.
ast	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.
stadd	The starting address of the data buffer. Stadd may be on a byte boundary.

Parameter	Meaning
size	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	The terminal driver responds to task I/O requests for cursor positioning without the task requiring information about the type of terminal in use. The vfc parameter specifies cursor position. The parameter is interpreted 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. 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. Table 2-9 is a summary of all 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.

Subfunction Bits:

The subfunctions have the following meanings:

Subfunction	Meaning
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. 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. When included in this manner, TF.RCU causes the driver to first save the current cursor position, then position the cursor and output the specified buffer, and, finally, restore the cursor to the original (saved) position once the output transfer has been completed.
TF.WAL	Write All Bits - During a write-pass-all operation (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 input/output wrap-around has been selected.

2.4.14 QIO$C SF.GMC - Get Multiple Characteristics

The SF.GMC QIO$ macro returns terminal characteristics information into a specified buffer. Table 2-4 in this section shows the terminal characteristics that can be obtained with this QIO$ macro.

The format of the QIO$C SF.GMC macro is as follows:

```
QIO$C SF.GMC,lun,[efn],[pri],[isb],[ast],<stadd,size>
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) 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 (ISB) is different from that 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.
ast	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.
stadd	The starting address of a data buffer of length "size" bytes. Each word in the buffer has the form .BYTE characteristic-name .BYTE 0 **characteristic-name** One of the bit names given in Table 2-5. The value returned in the high byte of each byte-pair is 1 if the characteristic is true for the terminal and 0 if it is not true.
size	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 2-5 is returned in the high byte.

Table 2-4
Terminal Characteristics
for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (if asserted)	DCL Command
TC.ABD	77	Auto-baud detection	SET TERM/AUTOBAUD
TC.ACD	---	Ancillary control driver. Value determined by system manager.	--
TC.ACR	24	Wrap-around mode	SET TERM/WRAP
TC.ANI	122	ANSI CRT terminal	SET TERM/ANSI_CRT
TC.ASP	76	Remote line answer speed. Initial speed over dial-up line.	SET TERM/REMOTE/SPEED:brate
TC.AVO	123	VT100-family terminal display	SET TERM/ADVANCED_VIDEO
TC.BIN	65	Binary input mode (read-pass-all). No characters are interpreted as control characters.	SET TERM/PASSALL
TC.BLK	42	Terminal is capable of block mode transfers	SET TERM/BLOCK_MODE
TC.CTS	72	Suspend output to terminal. Task can cancel an input ^S or get the current state of the terminal with regard to ^S or ^Q. 0 = resume 1 = suspend	--
TC.DEC	124	Digital CRT terminal	SET TERM/DEC_CRT
TC.DLU[1]	41	Dial-up line	SET TERM/REMOTE
TC.EDT	125	Terminal performs editing functions	SET TERM/EDIT_MODE

1. A program can enable the auto-call feature of the DF03 modem by setting TC.DLU to a value of two. Auto-call 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).

(continued on next page)

Table 2-4 (Cont.)
Terminal Characteristics
for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (if asserted)	DCL Command
TC.EPA	40	When TC.PAR is enabled: 0 = odd parity 1 = even parity	SET TERM/PARITY:value
TC.ESQ	35	Input escape sequence recognition	SET TERM/ESCAPE
TC.FDX	64	Full-duplex mode	SET TERM/FULL_DUPLEX
TC.HFF	17	Hardware form-feed capability (If 0, form-feeds are simulated using TC.LPP.)	SET TERM/FORM_FEED
TC.HFL	13	Number of fill characters to insert after a carriage return (0-7=x)(Use a value of 7 for the LA30-S.)	SET TERM/CRFILL
TC.HHT	21	Horizontal tab capability (if 0, horizontal tabs are simulated using spaces.)	SET TERM/TAB
TC.HLD	44	Hold screen mode. Terminal has ability to hold screen. Not supported over net (NCT).	SET TERM/HOLD_SCREEN
TC.HSY	137	Host to terminal synchronization. XOFF sent when resources are low. XON sent when resources are high. XOFF prevents terminal character input. 0 = No flow control 1 = Flow control exerted	SET TERM/HOST_SYNC
TC.ICS	141	Notify of change in type-ahead buffer (input count state)	
TC.ISL	6	Get MUX subline (=0-15) on interface to which user is connected (SF.GMC only).	--

(continued on next page)

Table 2-4 (Cont.)
Terminal Characteristics
for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (if asserted)	DCL Command
TC.LPP	2	Page length (1-255.=x)	SET TERM/PAGE_LENGTH:n
TC.MHU	145	Declare modem hangup AST. Specify address of AST activated by lost carrier.	--
TC.NBR	102	Broadcast disabled	SET TERM/NOBROADCAST
TC.NEC	47	Echo suppressed	SET TERM/NOECHO
TC.OOB	140	Specify out-of-band characters, whether they are included in the type-ahead buffer, and if they are to clear the type-ahead buffer.	--
TC.PAR	37	Generate and check parity	SET TERM/PARITY:value
TC.PPT	147	Terminal has printer port	SET TERM/PRINTER_PORT
TC.PRI	51	Terminal is privileged (SF.GMC only)	SET TERM/PRIVILEGED
TC.PTH	146	Pass through enable Only CTRL/S and CTRL/Q are honored. 1 = pass through 0 = default; no pass through	SET TERM/PASSTHRU
TC.RAT	7	Type-ahead buffer: 0 = 1-character type-ahead 1 = 36-character type-ahead (RSX-11M only)	SET /TYPEAHEAD=TTnn:
TC.RGS	126	Terminal supports REGIS instructions	SET TERM/REGIS
TC.RSP	3	Receiver speed (bits-per-second)	SET TERM/SPEED=:(xmit,rcv)
TC.SCP	12	Terminal is a scope (CRT)	SET TERM/SCOPE
TC.SFC	131	Terminal supports soft character set	SET TERM/SOFT_CHARACTERS
TC.SLV	50	No unsolicited input is accepted	SET TERM/SLAVE

(continued on next page)

Table 2-4 (Cont.)
Terminal Characteristics
for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (if asserted)	DCL Command
TC.SMR	25	Upper-case conversion disabled	SET TERM/UPPERCASE
TC.SSC	142	Specify terminal management switch characters. These cause a switch from normal mode to terminal management mode.	--
TC.TBF	71	Type-ahead buffer count obtained by SF.GMC. Cleared by SF.SMC.	--
TC.TBM	101	Type-ahead buffer mode 0=task type-ahead 1=CLI type-ahead	SET TERM/SERIAL
TC.TBS	100	Type-ahead buffer size (0-255=x) (RSX-11M-PLUS I/D systems only)	SET TERM/TYPEAHEAD:n
TC.TLC	130	CLI gets CTRL/C notification	SET TERM/CONTROL=
TC.TMM	143	In terminal management mode. Set when switch characters have been detected and terminal management mode is active. Cleared by QIO$ SF.SMC. 1 = In terminal management mode 0 = Exit terminal management mode	--
TC.TSY	144	Output flow control Allows input XON or XOFF to function. XOFF prevents output from the terminal. 0 = XON/XOFF ignored 1 = default; process XON/XOFF	SET TERM/TTSYNC
TC.TTP	10	Terminal type (=0-255.=x)	SET TERM/type
TC.VFL	14	Send four fill characters after line feed for vetical forms control.	SET TERM/LFILL=TTnn:

(continued on next page)

Table 2-4 (Cont.)
Terminal Characteristics
for SF.GMC and SF.SMC Functions

Bit Name	Octal Value	Corresponding Meaning (if asserted)	DCL Command
TC.WID [2]	1	Page width (=1-255.=x)	SET TERM/WIDTH:n
TC.XSP	4	Transmitter speed (bits-per-second)	SET TERM/SPEED:(xmit,rcv)
TC.8BC	67	Pass eight bits on input, even if not binary input mode (TC.BIN).	SET TERM/EIGHTBIT

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 echoing ^U.

In Table 2-5, the octal values 0-177 are reserved by DIGITAL. Values 200-377 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 2-5
Bit TC.TTP (Terminal Type) Values
Set by SF.SMC and Returned by SF.GMC

Octal Value	Symbol	Terminal Type	Implicit Characteristics					
			TC.LPP	TC.WID	TC.HFF	TC.HHT	TC.HFL	TC.VFL	TC.SCP
0	T.UNK0	Unknown							
15	T.V100	VT100	24	80		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			
34	T.LQP2	LQP02	66	132	1	1			
35	T.V2XX	VT2XX	24	80					
37	T.LN03	LN03	66	132	1	1			
40	T.DTC1	DTC01	66	132					

2.4.14.1 **Characteristic Bit Special Information** - The following bits have special, additional information:

- **TC.RSP, TC.XSP, TC.ASP** - The DCL SET TERM/SPEED command requires parameters for both receiver (rcv) and transmitter (xmit) baud rates. (The valid combinations for each are in the RSX-11M/M-PLUS Command Language Manual.) The DCL SET TERM/SPEED command format is as follows:

  ```
  SET TERM/SPEED:(xmit,rcv)
  ```

  The list of baud rates in bps and valid DCL SET TERM/SPEED or SET TERM/REMOTE values that may be set is as follows:

TC.ASP TC.RSP or TC.XSP Value	Baud Rate (in bps) And Valid DCL 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 (Not available on DHV11 multiplexer.)
S.4800	4800
S.7200	7200 (Not available on DHV11 multiplexer.)
S.9600	9600
S.19.2	19200 (Not available on DZV11 or DZQ11 multiplexer.)

  DZV11 and DZQ11 transmitter and receiver speeds must be equal (no split baud rates permitted). Only one value may be specified for the remote answer speed. This value applies to both the transmitter and 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.HHT, TC.VFL, and TC.SCP as shown in Table 2-4. You can change (override) these values by subsequent Set Multiple Characteristics requests. In addition, the terminal driver uses TC.TTP to determine cursor positioning commands, as appropriate.

- **TC.CTS** - Returns the current 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 if 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. or 132. 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 (QIO$ IO.ATT) the terminal to receive characters from the type-ahead buffer.

  2. The maximum capacity of the type-ahead buffer is 255(decimal) characters.

  3. Using TC.TBF in an SF.SMC function flushes the type-ahead buffer.

2.4.15 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 2-4 notes the restrictions that apply to these characteristics.

If the characteristic-name is TC.TTP (terminal type), value can have any of the values listed in Table 2-5.

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).

Before a remote line is answered the driver clears certain terminal characteristics that may have been set by an SF.SMC function.

For SF.SMC, the contents of the I/O Status Block (ISB) is different from that 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 successfully 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.

The format of QIO$C SF.SMC is as follows:

```
QIO$C SF.SMC,lun,[efn],[pri],[isb],[ast],<stadd,size>
```

Parameters:

The parameters have the following meanings:

Parameter	Meaning
lun	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	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	The address of the I/O status block (I/O status double-word) associated with the I/O request. For more information refer to Chapter 1; however, the I/O status block used for SF.SMC is different than that described in Chapter 1.
ast	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.

Parameter	Meaning
stadd	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, and TC.OOB, each word in the buffer has the form .BYTE characteristic-name .BYTE value **characteristic-name** One of the symbolic bit names given in Table 2-4. **value** Either 0 (to clear a given characteristic) or 1 (to set a characteristic).
size	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 must be an even value.

2.4.15.1 **Characteristic Processing for TC.MHU, TC.SSC, and TC.OOB** - Three characteristics require special processing and buffers. These characteristics are TC.MHU, TC.SSC, and TC.OOB. The buffers have the form

```
.BYTE characteristic name
.BYTE reserved
.WORD ...
```

The processing for the four characteristics is described following:

Characteristic	Processing
TC.MHU	This characteristic declares a modem hangup 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 following form:

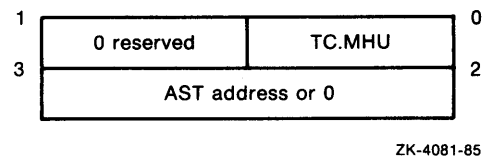


Figure 2-3 Buffer Required for TC.MHU

Characteristic	Processing

TC.SSC The characteristic TC.SSC defines and redefines terminal switch characters. The buffer required for TC.SSC is shown in Figure 2-4. Switch characters can be disabled while the terminal is in terminal management mode. The terminal must be attached (IO.ATT) before you set this characteristic. However, the terminal must not be attached for notification of unsolicited input ASTs (IO.ATA).

The buffer has the following format:

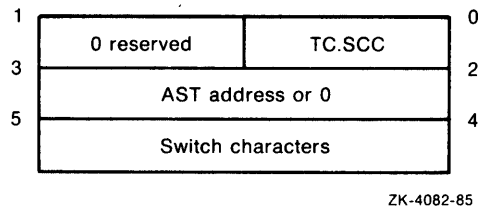


Figure 2-4 Buffer Required for TC.SSC

When the AST address is zero, 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, it 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. The buffer required for TC.OOB is shown in Figure 2-5. The terminal must be attached (IO.ATT) before you set this characteristic. However, the terminal must not be attached for notification of unsolicited input ASTs (IO.ATA).

The buffer has the following format:

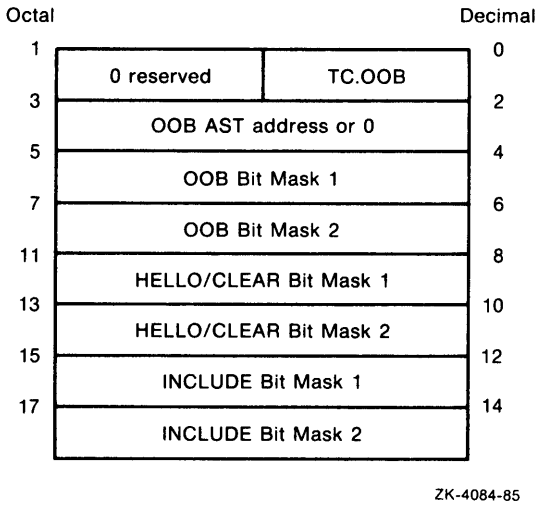


Figure 2-5 Buffer Required for TC.OOB

- Because all OOBs are either HELLO or CLEAR, one set of bit masks may be used for both. A zero bit mask is a CLEAR. A one 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 only the one.

2.4.15.2 **Side Effects of Setting Characteristics** - Certain terminal characteristics that a task may set or that an operator may set using DCL commands may have undesirable side effects. In particular, these characteristics include the hold-screen mode and the lowercase to uppercase conversion disable mode. Their effects are described following:

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 in the hold-screen mode, the terminal automatically sends a CTRL/S during reception of an output stream when 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 character, 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 are considered data characters.

It is advisable to specify nonordinary characters as switch characters, for example, non-system-specific CTRL-key combinations.

2.5 STATUS RETURNS

Table 2-6 lists error and status conditions that are returned by the terminal driver to the I/O status block.

Most Micro/RSX error and status codes returned are byte values in the status word. For example, the value for IS.SUC (success) is 1 and is placed in the low byte of the first status word. However, IS.CC, IS.CR, IS.ESC, and IS.ESQ are word values in the first word of the status block. They show 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 I/O status block for the value IS.SUC. Then, test the full word for IS.CC, IS.CR, IS.ESC, or IS.ESQ. (If the full word tests 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 characters returned to the task's buffer can be terminated by a CTRL/Z character.

The SE.xxx codes are returned by the SF.GMC and SF.SMC functions that are described in Sections 2.4.14 and 2.4.15. When any of these codes are returned, the low byte in the first word in the I/O status block contains IE.ABO. The second IOSB word contains an offset (starting from 0) to the byte in error in the QIO$'s stadd buffer.

Table 2-6
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 I/O status block 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 exceeds 8128 bytes.
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 result by pressing the BREAK key on some terminals, or by hardware problems.
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, not that the unit was attached by another task. If the attach specified TF.AST or TF.ESQ, these subfunction bits have no effect.

(continued on next page)

Table 2-6 (Cont.)
Terminal Status Returns

Code	Reason
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, 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 time-out 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 I/O status block contains the number of bytes read before CTRL/Z was seen. The input buffer contains those bytes.
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** 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.
IE.NOD	**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 AST block for an attach request.

(continued on next page)

Table 2-6 (Cont.)
Terminal Status Returns

Code	Reason
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, directed an SF.SMC to a terminal other than TI:, or it attempted to set its privilege bit.
IE.SPC	**Illegal address space** This error can indicate one or more of the following errors: • 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 a 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.

(continued on next page)

Table 2-6 (Cont.)
Terminal Status Returns

Code	Reason
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.
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 I/O status block is filled with 0s.
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 I/O status block 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 time-out (TF.TMO was set and the specified time interval was exceeded). The input buffer contains the bytes read.
SE.ATA	The terminal is attached with AST notification enabled.
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 in Table 2-4 was named in an SF.GMC or SF.SMC request, or a task attempted to assert TC.PRI.

(continued on next page)

Table 2-6 (Cont.)
Terminal Status Returns

Code	Reason
SE.NSC	An attempt was made to change a nonsettable characteristic. 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 was not enough room in pool 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 2-5.

2.6 CONTROL CHARACTERS AND SPECIAL KEYS

This section describes the meanings of special terminal control characters and keys for Micro/RSX. 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 control key (CTRL) down while typing one other key. Three of the control characters described in Table 2-7, CTRL/R, CTRL/U, and CTRL/Z, are echoed on the terminal as ^R, ^U, and ^Z, respectively.

Table 2-7
Terminal Control Characters

Character	Meaning
CTRL/C	If the command line interpreter (CLI) is DCL, which is the CLI for Micro/RSX, CTRL/C either aborts all currently active tasks invoked from the terminal or gives a **DCL>** prompt. To setup CTRL/C to abort, the terminal characteristics bit (TC.TLC) must be set. For more information refer to Section 2.6.2.
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. This character has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST or TF.RPT are invoked, and it is treated as a normal character.

(continued on next page)

Table 2-7 (Cont.)
Terminal Control Characters

Character	Meaning
CTRL/J	CTRL/J is equivalent to a LINE FEED character. This character has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST or TF.RPT are invoked, and it is treated as a normal 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. This character has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST or TF.RPT are invoked, and it is treated as a normal character.
CTRL/L	CTRL/L initiates a formfeed. If the terminal has hardware formfeed support, the driver echos CTRL/L. Otherwise, the driver simulates the formfeed 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. This character has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST or TF.RPT are invoked, and it is treated as a normal character.
CTRL/M	CTRL/M is equivalent to a RETURN character (see Section 2.6.3). This character has no special function if IO.RAL, IO.RST, TF.RAL, TF.RST or TF.RPT are invoked, and it is treated as a normal character.
CTRL/O	If the passthrough terminal characteristic has been set (TC.PTH), or if IO.RAL or TF.RAL is enabled CTRL/O is treated as a normal character. Otherwise, CTRL/O suppresses terminal output. In addition, 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 typed. • 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).

(continued on next page)

Table 2-7 (Cont.)
Terminal Control Characters

Character	Meaning
CTRL/Q	If IO.RAL or TF.RAL is enabled, CTRL/Q is treated as a normal character. Otherwise, CTRL/Q resumes terminal output previously suspended by means of CTRL/S only if the terminal characteristic TC.TSY has been enabled.
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 by setting the TC.TSY terminal characteristic bit.
CTRL/R	If TF.RNF, IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT are enabled, CTRL/R is treated as a normal character. Otherwise, typing CTRL/R results in a carriage return and line feed being echoed, followed by the incomplete (unprocessed) input line. Any tabs that were input are expanded and the effect of any rubouts is shown. On hardcopy terminals, CTRL/R allows verifying the effect of a tab or a rubout, or both, in an input line. CTRL/R allows verifying the effects of tabs or rubouts in an input line. For example, after rubbing out the left-most 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/U	If TF.RNF, IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT are enabled, CTRL/U is treated as a normal character. Otherwise, typing 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	If IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT are enabled, CTRL/X is treated as a normal character. Otherwise, CTRL/X clears the type-ahead buffer.
CTRL/Z	If IO.RAL, TF.RAL, IO.RST, TF.RST, or TF.RPT are enabled, CTRL/Z is treated as a normal character. Otherwise, CTRL/Z indicates an end-of-file for the current terminal input. It signals MAC, 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 CTRL/C Processing

The terminal driver can pass CTRL/C characters directly to the terminal's CLI for processing instead of doing the processing itself. Under these conditions, it is the CLI that is programmed to respond to a CTRL/C typed at a terminal. For instance, the DCL command line interpreter responds by aborting all currently active tasks invoked from the terminal. For the terminal driver to behave this way, both the CLI and the terminal must be set to handle CTRL/C characters. The terminal driver passes CTRL/C processing to a CLI only when the CLI is initialized to process CTRL/C notification packets and the terminal is set to trap CTRL/C characters.

To setup CLI CTRL/C processing, use the CTRLC switch in the CLI command as follows:

```
CLI /NAME=cliname/CTRLC
```

(This command sets CP.CTC in the CLI's CPB. Micro/RSX DCL is already initialized this way.)

To set a terminal to trap CTRL/C characters you must set the TC.TLC terminal characteristics bit. This done by a SET command as follows:

```
SET TERM/CONTROL=C
```

At the QIO level a SF.SMC QIO is used to set the TC.TLC bit.

If the CLI and the terminal are initialized to handle CTRL/C characters, the terminal driver flushes the buffer, checks for ASTs, and creates a CLI notification packet that is passed to the CLI on behalf of the terminal. (A CLI notification packet is not created if a CTRL/C AST occurs from a task attached to a terminal.) The terminal driver takes no other action. At this point the CTRL/C notification packet is available to the CLI like a regular command line. The CLI can identify and process it however it wants to.

If the CLI and terminal are not initialized to process CTRL/C characters, typing CTRL/C causes unsolicited input on that terminal to be directed to a control line interpreter task, such as DCL, if the terminal is not attached. (Command line interpreters are invoked and display a prompt in a manner similar to that of DCL; therefore, for the purposes of this discussion, it is assumed that DCL is the command line interpreter in use, although the terminal driver will respond to other command line interpreters in a similar manner.) The **DCL>** prompt is echoed when the terminal driver is ready to accept an unsolicited DCL command line for input. When the unsolicited input is terminated, the command line is passed to DCL.

If the last character typed on the terminal was a CTRL/S (suspend output), CTRL/C restarts suspended output and directs subsequent input to DCL.

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 2.4.2). CTRL/C characters are also passed to a task if an IO.RAL or IO.RST function is effected.

If the terminal driver receives a CTRL/C character during a read operation (except during a Read-Pass-All operation or a Read With Special Terminators operation), the read operation is terminated, the type-ahead buffer is cleared, and an IS.CC status code is returned to the task.

2.6.3 Special Keys

The ESCape, carriage RETURN (or RUBOUT) keys have special significance for terminal input, as described in Table 2-8. A line can be terminated by an ESCape (or ALTmode), carriage RETURN, or 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 examining Word 5 of the buffer. An operator can obtain the same information with the DCL SHOW TERM/width command.

Table 2-8
Special Terminal Keys

Key	Meaning
ESCape	If IO.RAL, TF.RAL, or TF.RPT are enabled, ESCape is treated as a normal character. Otherwise, if escape sequences are not recognized, typing ESCape or ALTmode signals 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, ESCape signals the beginning of an escape sequence. (See Section 2.7.)
RETURN	If IO.RAL, TF.RAL, or TF.RPT are enabled, RETURN is treated as a normal character. Otherwise, typing RETURN terminates the current line and causes the carriage or cursor to return to the first column on the line.
RUBOUT	If TF.RNF (read no filter) is enabled, RUBOUT is treated as a normal character. Otherwise, typing DELETE or RUBOUT 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 typing successive DELETEs or RUBOUTs. 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, 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

(continued on next page)

Table 2-8 (Cont.)
Special Terminal Keys

Key	Meaning
	The second backslash is not displayed if a line terminator is typed after rubbing out the characters on a line, as in the following example: ABC\CBA If the CRT rubout feature was selected, DELETE or RUBOUT 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 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 ESC (033) character. Some terminals generate an escape sequence when a special key is pressed. On any terminal, an escape sequence may be generated manually by typing ESCape 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 Standard X 3.41-- 1974 and used in the VT100 is:

ESC ... F

where:

ESC The introduced control character (33 octal) that is named escape.

... The intermediate bit combinations that may or may not be present. These characters are bit combination 40(octal) to 57(octal) inclusive in both 7- and 8-bit environments.

F The final character. F characters are bit combinations 60 (octal) to 176 (octal) inclusive in escape sequences in both 7- and 8-bit environments.

The occurrence of characters in the inclusive ranges 0(octal) to 37(octal) is technically an error condition. The recovery from this error occurs upon immediate execution of the function specified by the character and the continuation of the escape sequence execution. The exceptions are: if the character ESC occurs, the current escape sequence is aborted, and a new one commences, beginning with the ESC just received; if the character CAN 30 (octal) or the character SUB 32 (octal) occurs, the current escape sequence is aborted, as is the case with any control character.

There are five exceptions to this general 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 with the DCL SET command:

```
SET TERM/ESCAPE
```

After this prerequisite is satisfied, one of the following prerequisites must be met:

1. You must attach the terminal with IO.ATT!TF.ESQ.
2. You must use the TF.RES modifier with the IO.EIO!TF.RLB function.

NOTE

The second method enables escape 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 octal) 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 rubout 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. Neither are they recognized in a Read All (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. Typing it at any point within an escape sequence causes the entire sequence to be abandoned and deleted from the input buffer. Thus, use DELETE or RUBOUT to abandon an escape sequence, if desired, once you have begun it.

2.7.4.2 **Control Characters (0-037)** - The reception of any except four characters in the range 0 to 037. 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:

```
ESC CTRL/S A
```

gives:

IOSB

IS.ESQ
2

with the additional effect of turning off the output stream.

2.7.4.3 **Full Buffer** - A syntax error results when an escape sequence is terminated by running out of read-buffer space, rather than by receipt of 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:

```
ESC ! A
```

the buffer contains "ESC !", and the I/O status block contains:

IOSB

IE.PES
2

The "A" is treated as unsolicited input.

2.7.5 Exceptions to Escape Sequence Syntax

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(s) in use for correct use of escape sequences.

2.8 VERTICAL FORMAT CONTROL

Table 2-9 is a summary of all 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 IO.WLB, IO.WVB, IO.WBT, IO.CCO, or IO.RPR functions.

Table 2-9
Vertical Format Control Characters

Octal Value	Character	Meaning
040	blank	SINGLE SPACE - The driver 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 - The driver 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, the driver 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 - The driver prints the contents of the buffer and outputs a carriage return, normally overprinting the previous line.
044	$	PROMPTING OUTPUT - The driver outputs one line feed and prints the contents of the buffer. This mode of output is for use with a terminal on which a prompting message is output, and input is then read on the same line.
000	null	INTERNAL VERTICAL FORMAT - The driver 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

Individual terminals can be set for wrap-around, as desired, using the DCL SET command

```
$SET TERM/WRAP
```

Once wrap-around has been selected, the column at which wrap-around occurs can be selected using the DCL command

```
$SET TERM/WIDTH
$
```

The SHOW TERM/WIDTH command can display the current buffer width for a terminal:

```
$SHOW TERM/WIDTH
BUF=TI:00072.
$
```

A task can determine the buffer width by issuing a Get LUN Information directive and examining word 5 returned in the buffer.

After the SET has been done, typing beyond the buffer width results in a carriage return and line feed being output before the next character is echoed. Although only one line was input, it is displayed on two terminal lines.

It is possible to lose track of where you are in the input buffer if wrap-around 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.

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 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 error
IE.DAO	Data overrun
IE.VER	Character parity error

2. If a command line is being input for a command line interpreter 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 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 previously described is taken and the flag is cleared. Any characters received in the meantime are discarded, with a bell 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. 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 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, an attempt is made to retrieve a character from the buffer. If this 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 character is entered.
- Switch characters are detected.
- The terminal becomes detached.

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 simultaneously service one read request and one write request. The Attach, Detach and Set Multiple Characteristics 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, a second attempt is made in the system pool. If the allocation in the system pool fails during command line input, a CTRL/U is simulated and echoed.

Command line interpreter 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, secondary pool is allocated, the command line is copied into it, and it is queued to the CLI for processing.

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 I/O status block 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 DMA transfers for DHV11 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

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 as follows.

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 upon terminal type, the driver outputs appropriate 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 to first save the current cursor position, then position the cursor and output the specified buffer, and, finally, restore the cursor to the original (saved) position once the output transfer has been completed.

2.17 PROGRAMMING HINTS

2.17.1 Modem Support

The terminal driver supports the following modem control operations:

- Local or remote operation
- Answer speed
- Auto-baud speed detection

The characteristics bit that controls local or remote operation is TC.DLU. This bit can be set with the DCL command SET TERMINAL REMOTE (or SET TERMINAL LOCAL).

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 answer speed can be set on line using the DCL command SET/TERM/REMOTE/SPEED:brate.

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   1800
150   2400
300   4800
600   9600
1200
```

This is called auto-baud speed detection. This option can be selected for each line using the SET TERM/AUTOBAUD command. This command sets the TC.ABD terminal characteristic. When auto-baud speed detection is set for a given line, the terminal driver attempts to sense the baud speed of the caller when that line is set to remote and a call has been received. For auto-baud speed detection to work correctly, the user should input carriage returns when first establishing the remote connection until the CLI prompt is displayed on the screen.

2.17.2 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 stratagem such as issuing a read followed by a mark-time 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 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.

CHAPTER 3

DISK DRIVERS

3.1 INTRODUCTION

The Micro/RSX System has the following loadable disk drivers:

3.1.1 DUDRV.TSK

The DUDRV driver processes I/O requests for the RQDX1/RQDX2 controller. The RQDX1/RQDX2 controller is for the RD51-A and the RD52. The RD51 is a 10-megabyte fixed Winchester disk and the RX50-AA, a dual 400 kilobyte 5.25-inch diskette drive, both of which are included with a delivered Micro/RSX System. The RD52 is a 30.97-megabyte fixed Winchester disk.

The DUDRV driver also processes I/O requests for the RC25 fixed/removable disk system and the KDA50 controller.

3.1.1.1 **RC25 Disk Hardware Description** - 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.

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. A master-slave configuration would contain four logical units.

3.1.1.2 **KDA50 Controller Hardware Description** - The KDA 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 QIOs. Therefore, QIO$ macros may not complete in the order in which they were issued. Also, the KDA can carry out an extensive self-test on power-up or initialization. The types of drives that can be connected to the KDA50 controller are the RA60 disk drive, which has a removable pack, and the RA80 and RA81, both of which are fixed media drives.

3.1.2 DLDRV.TSK

The DLDRV driver processes I/O requests for the RLV12 controller. This controller can handle from one to four 10-megabyte RL02 disk drives. These drives take top-loading removable disk cartridges.

3.1.3 DYDRV.TSK

The DYDRV driver processes I/O requests for the RXV21 controller. This controller controls a tabletop diskette drive subsystem that has two RX02 disk drives. Each drive takes an 8-inch flexible diskette that has 512 kilobytes of storage.

3.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information macro (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	22-bit direct addressing supported
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. You should clear the high byte in the buffer before using the block number. For DUDRV 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.

3.3 QIO MACRO

This section summarizes the standard and the device-specific QIO functions for disk drivers.

3.3.1 Standard QIO Functions

Table 3-1 lists the standard functions of the QIO macro that are valid for disks.

Table 3-1
Standard QIO Functions for Disks

Format	Function
QIO$C IO.ATT,...	Attach device[1]
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Kill I/O[2]
QIO$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$C IO.WLC,...,<stadd,size,,blkh,blkl>	Write logical block followed by write check[3]
QIO$C IO.WVB,...,<stadd,size,,blkh,blkl>	Write virtual block

1. Only volumes mounted foreign may be attached. Any other attempt to attach a mounted volume will result in an IE.PRI status being returned in the I/O status doubleword.

2. 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.

3. Not supported on RX02 flexible disks.

stadd

The starting address of the data buffer (must be on a word boundary).

size

The data buffer size in bytes.

blkh/blkl

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 the low 16 bits.

IO.RVB and IO.WVB are associated with file operations (see the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual). For these functions to be executed, a file must be open on the specified 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 System 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 FDB for the end-of-file block number (F.EFBK) before closing the file. (Refer to the .EXTND routine description in the RSX-11M/M-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 invoke 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 user-specified retry algorithms for applications in which data reliability must be high.

The overlapped seek drivers for Micro/RSX support subfunction bit IQ.Q, which queues the request immediately without doing a seek (that is, uses implied seeks).

3.3.2 Device-Specific QIO Functions

The device-specific functions shown in Table 3-2 are valid only for the RX02 and the RL02 disk drives.

Table 3-2
Device-Specific Functions for the
RX02 and RL02 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) (RX02 only)
QIO$C IO.WPB,...,<stadd,size,,,pbn>	Write physical block

stadd The starting address of the data buffer (must be on a word boundary).

size The data buffer size in bytes must be even and greater than 0.

pbn The physical block number where the transfer starts (no validation will occur).

density The media density as follows:

0 = single density
2 = double density

3.3.3 Device-Specific QIO Function for DU: Devices

The DUDRV driver supports the device-specific QIO function shown in Table 3-3.

Table 3-3
Device-Specific QIO Function for the DUDRV Disk 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.	

3.4 STATUS RETURNS

The error and status conditions listed in Table 3-4 are returned by the disk drivers described in this chapter.

Table 3-4
Disk Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO macro was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.

(Continued on next page)

Table 3-4 (Cont.)
Disk Status Returns

Code	Reason
IS.PND	**I/O request pending** The operation specified in the QIO macro has not yet been executed. The I/O status block is filled with 0s.
IS.RDD	**Deleted data mark read** A deleted record was encountered during execution of an IO.RPB function. The second word of the I/O status block can be examined to determine the number of bytes processed (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 open** The task attempted to open a file on the physical device unit associated with specified LUN, but a file has already been opened by the issuing task on that LUN.
IE.BLK	**Illegal block number** An invalid logical block number was specified. IE.BLK would be returned if an attempt was made to write on the last track of an RL02 disk.
IE.BBE	**Bad block error** The disk sector (block) being read was marked as a bad block in the header word.
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.
IE.DNR	**Device not ready** The physical device unit specified in the QIO macro was not ready to perform the desired I/O operation.
IE.IFC	**Illegal function** A function code was specified in an I/O request that is invalid for disks.

(Continued on next page)

Table 3-4 (Cont.)
Disk Status Returns

Code	Reason
IE.NLN	**File not open** 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** 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 macro was not on line. When the system was booted, 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 macro 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.
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. Alternatively, a byte count of 0 was specified.
IE.VER	**Unrecoverable error** 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.

(Continued on next page)

Table 3-4 (Cont.)
Disk Status Returns

Code	Reason
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.

When a disk I/O error condition is detected, an error is usually not returned immediately. Instead, Micro/RSX attempts to recover from most errors by retrying the function as many as eight times. Unrecoverable errors are generally parity, timing, or other errors caused by a hardware malfunction.

3.5 POWER-FAIL RECOVERY

For DUDRV devices, the power-fail recovery is handled by the driver.

For DLDRV and DYDRV devices, a disk driver is called at its power-fail recovery entry point when a power-fail occurs. This recovery routine gives a disk sufficient time to spin back up and be ready to accept I/O requests.

There are three possible recovery procedures:

1. If a device is busy prior to the power-fail and the requested I/O has not completed, the routine times out and checks the device status until the device is ready for I/O operations to resume.

2. If a device is not busy prior to the power-fail and receives an I/O request after the power-fail, the routine times out and checks the device status until the device is ready for I/O operations to resume.

3. If a device is not busy prior to the power-fail and there are no pending I/O requests, operations resume with no special handling by the recovery routine.

Note in the first two cases that, if the maximum time-out count is reached before the disk spins back up and is ready, an unsuccessful I/O completion code is returned.

3.6 PROGRAMMING HINTS

3.6.1 RL02 Last Track Bad-Sector File

The driver write-protects the last track of an RL02 cartridge. This track contains the factory-recorded bad-sector file.

3.6.2 RX02 Media Characteristics

The RX02 device driver (DYDRV) dynamically updates the system data base to reflect the characteristics of the media in the RX02 drive. Recommended Action: User tasks should issue a QIO Sense Characteristics function before requesting the device's media characteristics with the GLUN$ macro.

3.6.3 Stall I/O for RC25 Disks

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 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 MOUNT command in the RSX-11M-PLUS Command Language Manual).

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.

3.6.4 Dismounting the RC25

You dismount a unit on the RC25 in the same way as for other disk devices, by using the DISMOUNT command. 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 4

TAPE DRIVERS

4.1 INTRODUCTION

The Micro/RSX system has three tape drivers: the DDDRV.TSK for the TU58 tape subsystem, the MSDRV.TSK for the TSV05 and TK25 tape subsystems, and the MUDRV.TSK for the TK50 tape subsystem.

4.2 MSDRV - TSV05/TK25 MAGNETIC TAPE

The MSDRV driver processes I/O requests for the TSV05/TK25 tape subsystem.

The TSV05 is a Q-bus device that reads and writes at 1600 bpi on a 1/2-inch 9-track tape at 25 inches/second in TS11 compatibility mode. 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 has comprehensive (internal) diagnostic test execution. Recording is 1600 bpi phase-encoded (PE).

The TK25 consists of a TKQ25 controller for the Q-bus and a TK25 streaming tape drive. The TK25 reads and writes data on a DC600A 1/4-inch tape cartridge that is recorded at 8K bpi on 10 serial data tracks in a serial serpentine recording method. The TK25 has a storage capacity of 60 Mbytes for 8 Kbyte data records, and it has a maximum data transfer rate of 55 Kbytes per second.

4.3 MUDRV - TK50 MAGNETIC TAPE

The TK50 is an integrated subsystem that consists of a controller for the Q-bus and a TK50 streaming tape drive. The controller handles all error recovery and correction, and it holds multiple outstanding commands in internal buffers. 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 and the storage capacity is approximately 100 Mbytes.

4.4 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information macro (the first characteristics word) contains the following information for TSV05/TK25 tape. 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, 1 if it is not)
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
8	1	22-bit direct addressing supported
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
15	0 or 1	Device mountable

Words 3 and 4 of the buffer are undefined; word 5 indicates the default buffer size, for magnetic tapes 512 bytes.

4.5 STANDARD QIO$ FUNCTIONS

Table 4-1 lists the standard functions of the QIO$ macro that are valid for supported magnetic tape devices.

Table 4-1
Standard QIO$ Functions for TSV05/TK25/TK50

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)

(Continued on next page)

Table 4-1 (Cont.)
Standard QIO$ Functions for TSV05/TK25/TK50

Format	Function
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)

stadd The starting address of the data buffer. It must be on a word boundary for MUDRV devices, and it may be on a byte boundary for MSDRV devices.

size The data buffer size in bytes. Size must be greater than 0, and, for a write, must be at least 14 bytes. For the TSV05/TK25 or TK50, data transfers may be odd or even.

IO.KIL terminates an I/O request that is in progress if any of the following occur:

- A select error (not applicable to TK50)
- Error recovery
- Interrupt servicing
- Device timeout servicing

4.6 DEVICE-SPECIFIC QIO$ FUNCTIONS

Table 4-2 lists the device-specific functions of the QIO$ macro that are valid for the supported magnetic tape devices. Additional details on certain functions appear as follows.

4.6.1 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.

4.6.2 IO.RWD

IO.RWD completion for MUDRV devices means that the rewind has been initiated. For MSDRV devices, IO.RWD completion means that rewind to BOT has been completed. Additional operations on that controller may then be queued by the driver until load point (BOT) is reached.

4.6.3 IO.RWU

IO.RWU is normally used 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.

Table 4-2
Device-Specific QIO$ Functions for Supported Tape Devices

Format	Function
QIO$C IO.DSE,...	Data Security Erase (TK50 only)
QIO$C IO.EOF,...	Write end-of-file mark (tape mark)
QIO$C IO.ERS,...	Erase
QIO$C IO.RLV,...,<stadd,size>	Read logical block reverse
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, 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

cb The characteristic bits to set.

nbs The number of blocks to space past (positive if forward, negative if reverse).

nes The number of EOF marks to space past (positive if forward, negative if reverse).

size The size of the stadd data buffer in bytes. Size must be greater than zero, and, for a write, must be at least 14 bytes. For MSDRV or MUDRV devices, data transfers may be odd or even.

stadd The starting address of the data buffer. It must be on a word boundary for MUDRV devices, and it may be on a byte boundary for MSDRV devices.

4.6.4 IO.ERS

Erases tape to provide an extended interrecord gap.

4.6.5 IO.DSE

Causes the TK50 to erase from the current position to end-of-tape and then rewind the tape to beginning-of-tape.

4.6.6 IO.SEC

This function returns 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	Reserved.	
1	Swap byte mode (read/write). For TSV05/TK25	X
2	Reserved	
3	Even parity (default is odd). (Not selectable for the TSV05, TK25, TK50.)	X
4	Tape is past EOT.	
5	Last tape command encountered EOF (unless last command was backspace).	
6	Writing is prohibited.	X
7	Writing with extended inter-record gap is prohibited (that is, no recovery is attempted after write error).	X
8	Select error on unit (not applicable to the TK50).	
9	Unit is rewinding.	
10	Tape is physically write-locked.	
11	1600 bpi density (for TSV05/TK25, bit 11=1) IO.SMO or IO.STC cannot modify bit 11.	
12	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 user).	

4.6.7 IO.SMO

This function can be used 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 4-3).

The IO.SMO function should be used 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.

4.7 STATUS RETURNS

The error and status conditions listed in Table 4-3 are returned by the MSDRV and MUDRV driver.

Table 4-3
MSDRV/MUDRV Tape Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO$ macro was completed successfully. The second word of the I/O status block 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$ macro has not yet been completed. The I/O status block is filled with 0s.
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.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, not that the unit was attached by another task.

(Continued on next page)

Table 4-3 (Cont.)
MSDRV/MUDRV Tape Status Returns

Code	Reason
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 transfered to memory.
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$ macro was not ready to perform the desired I/O operation. This code is returned to indicate one of the following conditions: • A time-out occurred on the physical device unit (that is, an interrupt was lost). • The physical drive was not on-line.
IE.EOF	**End-of-file was encountered.**
IE.EOT	**End-of-tape encountered.** The end-of-tape (physical end of tape (EOT) marker) was encountered while the tape was moving in the forward direction for a write or write tape mark operation. The IE.EOT code is returned continually in the I/O status block until the EOT marker is passed in the reverse direction. IE.EOT is not returned on a read operation. A length of tape is provided past EOT to be used for writing data and markers, such as volume trailer labels. For MUDRV devices, this length of tape must not be less than the length required to store the aggregate of the following records: • Two device dependent "maximum recommended record length" records • Three 80-byte records • Three tape marks The IE.EOT code is returned continually in the I/O status block until the EOT marker is passed in the reverse direction. IE.EOT is not returned on a read operation.

(Continued on next page)

Table 4-3 (Cont.)
MSDRV/MUDRV Tape Status Returns

Code	Reason
IE.EOV	**End-of-volume encountered (unlabeled tape)** On a forward space function, the logical end-of-volume was encountered. An end-of-volume 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, magnetic tape unit not ready and/or tape not at load point when IO.SMO is issued.
IE.IFC	**Illegal function** An invalid function (or subfunction bit) was specified in a magnetic tape I/O request. Refer also to Section 4.2.4.3.
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO$ macro 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 the user attempted to write a block that was less than 14 bytes long.
IE.VER	**Unrecoverable error** The operation could not be completed due to a data transfer error and unsuccessful error recovery attempts.
IE.WLK	**Write-locked device** The task attempted to write on a 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 4-3.

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 4-4. For all other cases the contents of this word is undefined.

Table 4-4
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
IO.RLV	Number of bytes transferred	Number of bytes in tape record
IO.RVB	Number of bytes transferred	Number of bytes in tape record
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	

4.7.0.1 **Select Recovery** - If a request fails because the desired unit is off line, no drive has the desired unit number, or the drive has its power off, the following message is output on the operator's console:

```
*** MSn: -- SELECT ERROR
```

n

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 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 offline, the drive positions the tape at BOT. This condition is treated as tape position lost or powerfail.

4.8 RETRY PROCEDURES FOR READS AND WRITES

For MUDRV devices (TK50), the controller handles all error correction and recovery.

For MSDRV devices, if a write operation fails, the driver attempts the following error recovery procedure a predetermined number of times:

1. Repositions the tape
2. Erases the tape to create an extended interrecord gap
3. Retries the write operation

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.

For MSDRV devices, if a read operation fails, the driver rereads the block in error a predetermined number of times.

4.9 POWER-FAIL RECOVERY FOR MSDRV/MUDRV TAPES

If a power failure or loss of physical control of the tape occurs, tape position is lost. (Note that an initial system boot simulates a recovery from a power failure.) Additionally, on auto-load drives, the tape will be positioned at BOT when the unit is turned on line.

To prevent accidental destruction of data currently on tape, the driver maintains a power-fail status indicator. When this indicator is set, the driver disallows any data transfer or tape motion commands until a successful rewind (IO.RWD), rewind unload (IO.RWU), or mount and set characteristics (IO.SMO) function is issued. The successful completion of these functions clears the power-fail indicator and all tape motion functions are allowed to proceed. It is also possible to issue the set and sense characteristics functions (IO.STC and IO.SEC) while the power-fail indicator is set. These functions, however, will 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 will be issued, thereby clearing the power-fail indicator.

4.10 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users of the MSDRV or MUDRV driver described in this chapter.

4.10.1 Block Size

Each block must contain at least 14 bytes for a write and may contain a maximum of 65,534 bytes. However, tape usage is more efficient with a larger buffer.

4.10.2 Importance of Resetting Tape Characteristics

A task that uses 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.

4.10.3 Aborting a Task

If a task is aborted while waiting for a tape unit to be selected, the MSDRV driver recognizes this fact within one second.

If you abort a task while it waits for a tape unit to complete a space operation, the MSDRV driver may allow spacing to the next tape mark.

For MUDRV devices (TK50), if you abort a task while it waits for a magnetic tape unit to complete a space operation, the driver may have spaced some or all of the requested spacing operation.

The MUDRV terminates the spacing function immediately, which leaves the tape at an indeterminate position. A rewind should be performed immediately with another spacing function to get to a known position on the tape.

4.10.4 End-of-Volume Status (Unlabeled Tape)

The MSDRV/MUDRV driver detects end-of-volume when it spaces over the second of two consecutive tape marks. The tape is left positioned between the two tape marks.

The 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 end-of-volume, the driver treats beginning of tape (BOT) as a tape mark. Therefore, any forward space operation from BOT that immediately encounters a tape mark will return IE.EOV.

If a space operation stops between two tape marks but does not space over the second one, the driver will return end-of-file rather than end-of-volume. Any subsequent space operation from this point that immediately spaces over the second tape mark will return end-of-volume.

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 end of volume.

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 end-of-volume on read operations, your application must keep track of the tape marks. The magnetic tape driver does not support two consecutive tape marks as end-of-volume on read operations.

4.10.5 Resetting Tape Transport Status or VCK

When 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 IO.RWD or IO.SMO QIO$ or including /RW or /REW switches to command requests (such as DMP).

4.10.6 Issuing QIOs

Users issuing QIOs 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 off-line to on-line or vice versa, 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 are:
  - 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 a write operation.
  - 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 and, therefore, the physical end-of-tape may be reached before all requests are processed. Thus, with multiple QIOs 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 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. This length must be long enough to store the aggregate of the following records:

- Two device dependent "maximum recommended record length" records
- Three 80-byte records
- Three tape marks

4.11 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 MOUNT command. These conditions where this occurs are as follows:

- The tape is mounted /NOLABEL
- The MOUNT command specifies an odd record size
- The MOUNT command specifies an odd block size

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.

4.12 DDDRV - TU58 CARTRIDGE TAPE

The DDDRV driver processes I/O requests for the TU58 tape subsystem. The TU58 is a tabletop dual tape drive that takes 256 kilobyte cartridge tapes. This tape drive is connected to the Micro/RSX System through one of the asynchronous serial lines.

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 using 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 in order to report a soft error message to the error logger.

4.12.1 Get LUN Information Macro

Word 2 of the buffer filled by the Get LUN Information macro (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

Bit	Setting	Meaning
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	22-bit direct addressing supported
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(decimal) blocks. Word 5 indicates the default buffer size, which is 512(decimal) bytes.

4.12.2 Standard QIO$ Functions

Table 4-5 lists the standard QIO$ macro functions of the QIO$ macro that are valid for the TU58.

Table 4-5
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.

stadd The starting address of the data buffer (must be on a word boundary).

size The data buffer size in bytes (must be even and greater than 0). For a write, the size must be at least 14 bytes.

lbn The logical block number on the cartridge tape where the data transfer starts (must be in the range of 0-777).

4.12.3 Device-Specific QIO$ Functions

The device-specific QIO$ macro functions that are valid for the TU58 are shown in Table 4-6.

Table 4-6
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

stadd

The starting address of the data buffer (must be on a word boundary).

size

The data buffer size in bytes (must be even and greater than 0).

lbn

The logical block number on the cartridge tape where the data transfer starts (must be in the range of 0-777).

Additional details for device-specific QIO$ functions are provided in the following paragraphs.

4.12.3.1 **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.

4.12.3.2 **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 insure data read reliability.

4.12.3.3 **IO.BLS** - The IO.BLS function is used for diagnostic purposes to position the tape to the specified logical block number. If you specify IO.BLS, you must use the IQ.UMD subfunction (see Chapter 1).

4.12.3.4 **IO.DGN** - The IO.DGN function is used 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. If you specify IO.DGN, you must use the IQ.UMD subfunction (see Chapter 1).

4.12.4 Status Returns

Table 4-7 lists the error and status conditions that are returned by the DDDRV driver for the TU58.

Table 4-7
TU58 Driver Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO$ macro was completed successfully. The second word of the I/O status block 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$ macro was not ready to perform the desired I/O operation.
IE.IFC	**Illegal function** A function code was specified in an I/O request that is invalid for the TU58.
IE.FHE	**Fatal hardware error**
IE.TMO	**Time-out error** The TU58 failed to respond to a function within the normal time specified by the driver.
IE.VER	**Unrecoverable error** After the system's standard number of retries (eight) had 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.

4.13 PROGRAMMING HINTS

4.13.1 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 MOUNT command. This occurs under the following conditions:

- The tape is mounted /NOLABEL
- The mount command specifies an odd record size
- The mount command specifies an odd block size

FCS adds the padding character, an octal 136 (^) circumflex, 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 5

LINE PRINTER DRIVER

5.1 INTRODUCTION

The Micro/RSX system line printer driver is LPDRV.TSK. LPDRV.TSK supports the following line printers:

- LP25 Line Printer

  The LP25 is a band printer that has a 285 line per minute capacity. The LP25 has a standard 64-character set and a 132-column format.

- LP26 Line Printer

  The LP26 is a band printer capable of 600 lines per minute. The LP26 has a standard 64-character set and a 132-column format.

- LN01 Laser Printer

  The LN01 is a non-impact 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 per minute, which equates to 600 lines per minute. Contributing to the high print quality is a printer resolution of 300 by 300 dots per inch. The LN01 offers the speed of a line printer with the advantages of a phototypeset device.

5.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information macro (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

Bit	Setting	Meaning
6	0	Mass-storage device
7	0	User-mode diagnostics supported
8	0	22-bit direct addressing supported
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, for line printers the width of the printer carriage (that is, 80 or 132).

5.3 QIO MACRO

Table 5-1 lists the standard functions of the QIO macro that are valid for line printers.

Table 5-1
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)

stadd The starting address of the data buffer (may be on a byte boundary).

size The data buffer size in bytes (must be greater than 0).

vfc A vertical format control character from Table 5-3.

IO.KIL does not cancel an in-progress request unless the line printer is in an off-line 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.

5.4 STATUS RETURNS

Table 5-2 lists the error and status conditions that are returned by the line printer driver described in this chapter.

Table 5-2
Line Printer Status Returns

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO macro was completed successfully. The second word of the I/O status block 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 macro has not yet been executed. The I/O status block is filled with 0s.
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, not that the unit was attached by another task.
IE.DNA	**Device not attached** The physical device unit specified that 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** An I/O request specified a function code that is illegal for line printers.

(Continued on next page)

Table 5-2 (Cont.)
Line Printer Status Returns

Code	Reason
IE.OFL	**Device off line** The physical device unit associated with the LUN specified in the QIO macro 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.

5.4.1 Ready Recovery

If any of the following conditions occur:

- Paper jam
- Printer out of paper
- Printer turned off line
- Power failure

the driver determines that the line printer is off line, and the following message is output on the operator's console:

```
***LPn: -- NOT READY
```

where:

n 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 every 15 seconds until the line printer is readied. If a power failure occurs while printing a line, the entire line is reprinted from the beginning when power is restored.

5.5 VERTICAL FORMAT CONTROL

Table 5-3 summarizes the meaning of all characters used 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 5-3
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 per I/O request.

All other vertical format control characters are interpreted as blanks (040(octal)).

5.6 PROGRAMMING HINTS

This section contains information on important programming considerations relevant to users of the line printer driver described in this chapter.

5.6.1 RUBOUT Character

The line printer driver discards the ASCII character code 177 during output.

5.6.2 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. The user can determine that truncation will occur by issuing a Get LUN Information macro and examining word 5 of the information buffer. This word contains the width of the print carriage in bytes.

5.6.3 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 one second.

CHAPTER 6

VIRTUAL TERMINAL DRIVER

6.1 INTRODUCTION

The virtual terminal driver supports offspring task use of virtual terminals in 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 Micro/RSX Executive. Virtual terminals are created by the Executive when requested by parent tasks with the Create Virtual Terminal macro. Virtual terminals are useful in batch processing and other processing environments in providing noninteractive terminal I/O support for offspring tasks, eliminating the need for operator intervention.

Offspring task(s) "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.

6.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information macro (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
5	0	Sequential device
6	0	Reserved
7	0	User-mode diagnostics supported
8	0	Massbus device
9	0	Unit software write-locked

Bit	Setting	Meaning
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/M-PLUS and Micro/RSX Executive Reference Manual.

6.3 QIO MACRO

Table 6-1 lists the standard and device-specific functions of the QIO macro that are valid for virtual terminals.

Table 6-1
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$C IO.WVB,...,<stadd,size,stat>	Write virtual block (effects IO.WLB)
DEVICE-SPECIFIC FUNCTIONS:	
QIO$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

(Continued on next page)

Table 6-1 (Cont.)
Standard and Device-Specific QIO Functions for Virtual Terminals

Format	Function
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

size 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 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 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 Characteristic bits to become 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

swl The I/O completion code for I/O completion status.

NOTE

The sw2 and swl parameters are valid in the IO.STC function only when cb=1 or cb=3.

tmo An optional time-out count (see following).

vfc A character for vertical format control.

pradd The starting address of the prompt buffer.

prsize 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.

6.3.1 Standard QIO Functions

6.3.1.1 **IO.ATT** - This I/O 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.

6.3.1.2 **IO.DET** - This I/O 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.)

6.3.1.3 **IO.KIL** - Parent and offspring tasks can issue IO.KIL to cancel I/O requests. An offspring task issuing IO.KIL can result in IE.ABO being returned to the parent task.

6.3.1.4 **IO.RLB, IO.RVB, IO.WLB, IO.WVB** - These read and write functions execute requested I/O operations with virtual terminals in the same manner as with terminals described in Chapter 2, except as follows:

1. The virtual terminal driver returns the tmo parameter of an offspring task's IO.RLB or IO.RVB request, or the vfc parameter of an offspring task's IO.WLB or IO.WVB request as a stack parameter on entry to the appropriate 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.

6.3.2 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 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 as follows.

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 that when physical terminals are used. Whenever the virtual terminal driver determines that intermediate buffering should not be used, 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 automatically 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(decimal) 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. 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 used in this manner, the IO.STC function must include three parameters, <cb,sw2,sw1>, as follows:

Parameter — **Meaning**

cb — A value of 1 is specified to indicate that the I/O completion status return 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 — This parameter is the second word returned in the I/O completion status indicating the number of bytes read upon successful completion of an offspring task's read request. However, since 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 will result in an error being returned to the parent task).

sw1 — This parameter 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 represent 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

6.3.3 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 6-2.

6.3.4 IO.GTS

The Get Terminal Support function returns a 4-word buffer of information specifying 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 2-5 lists the options returned by the full duplex terminal driver. Of those 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

6.3.5 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.

6.3.6 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. In order to insure that these operations are overlapped, the parent task should minimize the amount of time it spends in AST state.

The virtual terminal driver defaults to half duplex mode.

Table 6-2 lists the characteristics that either the parent or the offspring task may set.

Table 6-2
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

6.4 STATUS RETURNS

The error and status conditions listed in Tables 6-3 and 6-4 are returned by the virtual terminal driver described in this chapter. 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 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 6-3
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 drivers described in Chapter 2.
IS.SUC	**Successful completion** The operation specified in the QIO macro was completed successfully. The second word of the I/O status block 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.

(Continued on next page)

Table 6-3 (Cont.)
Virtual Terminal Status Returns for Offspring Task Requests

Code	Reason
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.
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 in Table 6-2 was specified, or an offspring task attempted to assert TC.FDX.

Table 6-4
Virtual Terminal Status Returns for Parent Task Requests

Code	Reason
IS.SUC	**Successful completion** The operation specified in the QIO macro was completed successfully. The second word of the I/O status block 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.

(Continued on next page)

Table 6-4 (Cont.)
Virtual Terminal Status Returns for Parent Task Requests

Code	Reason
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 in Table 6-2 was specified in an SF.GMC or SF.SMC request.

CHAPTER 7

NULL DEVICE DRIVER

Micro/RSX provides 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 error (IE.EOF).
- A write to NL: immediately returns success (IS.SUC).

The null device functions as a "wastebasket" to which you can direct output, and from which it will never return. It is particularly useful when used in conjunction with an indirect command file and MCR ASN commands, as in the example following.

Figure 7-1 shows the contents of a Task Builder command file called TESTBLD.CMD. Symbolic device names are used for the output file, map file, and input file. These names may be reassigned at task-build time. In particular, in the example following, the map file is assigned to the null device and thus is thrown away.

```
>ASN SY:=OU:

>ASN NL:=MP:

>ASN DK1:=IN:

>TKB @TESTBLD
```

```
OU:TEST,MP:TEST=IN:[200,220]TEST
/
ASG=TI:2
//
```

ZK-4085-85

Figure 7-1 Indirect TKB Command File TESTBLD.CMD.

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 Section 1.5.1. 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 TU58 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.2 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 - RX02 only)

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.3 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.4 MAGNETIC TAPE DRIVER

IO.ATT,...	Attach device
IO.DET,...	Detach device
IO.DSE,...	Data security erase (TK50 only)
IO.EOF,...	Write end-of-file (tape mark)
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.5 TERMINAL DRIVER

IO.ATA,...,<ast,[parameter2],[ast2]>	ATTACH device, specify unsolicited-character AST[1]
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[2]
IO.EIO!TF.WLB,...,<stadd,size>	Extended I/O Write Functions[2]
IO.GTS,...,<stadd,size>	Get terminal support
IO.HNG,...	Hangup remote line
IO.KIL,...	Cancel I/O requests
IO.RAL,...,<stadd,size,[tmo]>	Read logical block and pass all characters[1]
IO.RLB,...,<stadd,size,[tmo]>	Read logical block[1]
IO.RNE,...,<stadd,size,[tmo]>	Read logical block and do not echo[1]
IO.RPR,...,<stadd,size,[tmo],pradd,prsize,vfc>	Read after prompt[1]
IO.RST,...,<stadd,size,[tmo]>	Read with special terminators
IO.RTT,...,<stadd,size,[tmo],table>	Read logical block ended by specified special terminator[2]
IO.RVB,...,<stadd,size,[tmo]>	Read virtual block[1]
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[1]

TF.RAL	Read, pass all characters
TF.RCU	Restore cursor position[1]
TF.RDI	Read with default input (IO.EIO function only)[2]
TF.RES	Read with escape sequence processsing enabled (IO.EIO function only)[2]
TF.RLB	Read logical block (IO.EIO function only)[2]
TF.RLU	Read and convert from lower- to upper-case (IO.EIO function only)[2]
TF.RNE	Read with no echo
TF.RNF	Read with no filter (IO.EIO function only)[2]
TF.RPR	Read after prompt (IO.EIO function only)[2]
TF.RPT	Read in pass-through mode (IO.EIO function only)[2]
TF.RST	Read with special terminators
TF.RTT	Read with specified special terminator table (IO.EIO function only)[2]
TF.TMO	Read with time-out[1]
TF.WAL	Write, pass all bits
TF.WBT	Break-through write
TF.WIR	Write with input redisplayed
TF.XCC	CTRL/C starts a command line interpreter[1]
TF.XOF	Send XOFF

1. "ast2", "parameter2", and "tmo" parameters are available for full-duplex driver functions only.
2. Full-duplex driver only.

A.6 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. 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 STATUS CODES

This section lists error and success codes which can be returned in the I/O status block on completion of an I/O function. The codes below may be referenced symbolically by invoking the system macro IOERR$.

B.1.1 I/O Status Error 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
IE.ALN	-34	177736	File already open
IE.BAD	-01	177777	Bad parameter
IE.BBE	-56	177710	Bad block on device
IE.BCC	-66	177676	Block check error or framing error

Name	Decimal	Octal	Meaning
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 properly closed
IE.CNR	-96	177640	Connection rejected
IE.CON	-22	177752	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
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

Name	Decimal	Octal	Meaning
IE.FHE	-59	177705	Fatal hardware error
IE.FLG	-89	177647	Event flag already specified
IE.FLN	-81	177657	Device already offline
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
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 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
IE.NSF	-26	177746	No such file
IE.NST	-80	177660	Specified task not installed

Name	Decimal	Octal	Meaning
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	Sharable 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 (FCS or FCP)
IE.SZE	-98	177636	Unable to size device
IE.TML	-93	177643	Too many links to task
IE.TMO	-95	077641	Time-out on request
IE.UKN	-97	177637	Unknown name
IE.ULK	-85	177653	Unlock error
IE.URJ	-73	177667	Connection rejected by user
IE.VER	-04	177774	Parity error on device
IE.WAC	-29	177743	Accessed for write

Name	Decimal	Octal	Meaning
IE.WAT	-31	177741	Atribute 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	Decimal Bytes	Octal Word	Meaning
IS.CR	Byte 0: 1 Byte 1: 15	006401	Successful completion with carriage return
IS.CC	Byte 0: 1 Byte 1: 3	001401	Successful completion on read terminated by CTRL/C
IS.ESC	Byte 0: 1 Byte 1: 33	015401	Successful completion with ESCape
IS.ESQ	Byte 0: 1 Byte 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 time-out
IS.TNC	+02	000002	Successful transfer but message truncated (receiver buffer too small)

B.2 DIRECTIVE CODES

This section lists error and success codes that can be returned in the directive status word 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

Name	Decimal	Octal	Meaning
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.)
IE.IDU	-92	177644	Invalid device or unit
IE.IEF	-97	177637	Invalid event flag (.GT. 64.)
IE.ILU	-96	177640	Invalid logical unit number
IE.ILV	-19	177755	Invaild 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.)
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 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	Privileged violation
IE.RBS	-15	177761	Receive buffer is too small
IE.RSU	-17	177757	Resource in use
IE.SDP	-99	177635	Invalid DIC number or 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 standard and device-dependent I/O functions.

B.3.1 Standard 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 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 A/D conversion

B.3.3 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.3.4 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

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.SPB	002420	5	20	Space blocks
IO.SPF	002440	5	40	Space files

B.3.5 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.3.6 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.3.7 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

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.BLS	004010	10	10	Position tape
IO.DGN	004150	10	150	Run internal diagnostics

B.3.8 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, 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 only)
IO.WDD	001140	1	140	Write physical block with deleted data mark (RX02 only)
IO.WLC	001020	1	20	Write logical block followed by write check (all except RX01, RX02)
IO.WPB	000440	1	40	Write physical block (RX01, RX02, RL01, RL02 only)

B.3.9 Specific Graphics Display I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CON	015400	33	00	Connect to graphics device
IO.CNT	017000	36	00	Continue DPU
IO.DIS	016000	34	00	Disconnect from graphics device
IO.STP	016400	35	00	Stop DPU

B.3.10 Specific ICS/ICR, DSS/DR I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CCI	014000	30	0	Connect a buffer to digital interrupt input
IO.CTI	015400	33	0	Connect a counter
IO.CTY	003400	7	0	Connect a remote terminal
IO.DCI	014400	31	0	Disconnect a buffer from digital interrupt input
IO.DTI	016000	34	0	Disconnect a buffer from counter input
IO.DTY	006400	15	0	Disconnect a buffer from terminal input
IO.FLN	012400	25	0	Place selected unit off line
IO.ITI	017000	36	0	Initialize a counter
IO.LDI	007000	16	0	Link a task to digital interrupts
IO.LKE	012000	24	0	Link a task to error interrupts
IO.LTI	007400	17	0	Link a task to counter interrupts
IO.LTY	010000	20	0	Link a task to terminal interrupts
IO.MLO	006000	14	0	Open or close bistable digital output points
IO.MSO	005000	12	0	Pulse single-shot digital output points
IO.NLK	011400	23	0	Unlink a task from all unsolicited interrupts
IO.ONL	017400	37	0	Place selected unit on line
IO.RAD	010400	21	0	Read task activation data
IO.RBC	003000	6	0	Initiate multiple A/D conversions

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.SAO	004000	10	0	Perform analog output to specified channel
IO.UDI	011410	23	10	Unlink a task from digital interrupts
IO.UER	011440	23	40	Unlink a task from error interrupts
IO.UTI	011420	23	20	Unlink a task from counter interrupts
IO.UTY	011430	23	30	Unlink a task from terminal interrupts
IO.WLB	000400	1	0	Output to remote terminal

B.3.11 Specific LPA11-K I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CLK	015000	32	0	Start clock
IO.INI	014400	31	0	Initialize LPA11-K
IO.LOD	014000	30	0	Load microcode
IO.STA	015400	33	1	Start transfer
IO.STP	016400	35	0	Stop request

B.3.12 Specific LPS I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.ADS	014000	30	0	Initialize A/D sampling
IO.HIS	015000	32	0	Initialize histogram sampling
IO.LED	012000	24	0	Display number in LED lights
IO.MDA	016000	34	0	Initialize D/A output
IO.MDI	014400	31	0	Initialize digital input sampling
IO.MDO	015400	33	0	Initialize digital output

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.REL	013400	27	0	Latch output relay
IO.SDI	013000	26	0	Read digital input register
IO.SDO	012400	25	0	Write digital output register
IO.STP	016400	35	0	Stop in-progress request

B.3.13 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 only)
IO.EOF	003000	6	0	Write end-of-file gap
IO.RLV	001100	2	100	Read logical block (reverse)
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.3.14 Specific Parallel Communications Link I/O Function Codes - RSX-11M-PLUS Only

B.3.14.1 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.3.14.2 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.3.15 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	HANGUP 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 on-going I/O
SF.GMC	002560	5	160	Get multiple characteristics
SF.SMC	002440	5	40	Set multiple characteristics

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.RLB	000002
TF.RCU[1]	000001	TF.RLU[2]	000010
TF.CCO[1]	000040	TF.RTT[2]	000400
TF.WAL[1]	000010	TF.RST[2]	000001
TF.WBT[1]	000100	TF.BIN[2]	000002
TF.WIR[1]	000200	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

1. Modifiers of the IO.EIO!TF.WLB subfunction. These are specified by you in the item-list buffer.

2. Modifiers of the IO.EIO!TF.RLB subfunction. These are specified by you in the item-list buffer.

B.3.16 Specific UDC I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CCI	014000	30	0	Connect a buffer to contact interrupt digital input
IO.CTI	015400	33	0	Connect a timer
IO.DCI	014400	31	0	Disconnect a buffer from contact interrupt digital input

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.DTI	016000	34	0	Disconnect a timer
IO.ITI	017000	36	0	Initialize a timer
IO.MLO	006000	14	0	Open or close latching digital output points
IO.RBC	003000	6	0	Initiate multiple A/D conversions

B.3.17 Specific UNIBUS Switch I/O Function Codes - RSX-11M-PLUS Only

Symbolic Name	Word Equivalent	Code (High Byte)	Subcode (Low Byte)	Meaning
IO.CON	15400	33	0	Connect UNIBUS switch
IO.DIS	16000	34	0	Disconnect UNIBUS switch
IO.DPT	16010	34	10	Disconnect UNIBUS switch and connect to specified CPU port
IO.SWI	16400	35	0	Switch UNIBUS from current CPU to specified CPU
IO.CSR	15000	32	0	Read UNIBUS switch CSR

B.3.18 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

APPENDIX C

QIO$ INTERFACE TO THE ACPS

This appendix describes the QIO$ level interface to the file processors (ACPs). These include F11ACP for Files-11 disks and MTAACP for ANSI magnetic tape.

F11ACP supports the following functions:

IO.CRE	Create file
IO.DEL	Delete file
IO.ACR	Access file for read only
IO.ACW	Access file for read/write
IO.ACE	Access file for read/write/extend
IO.DAC	Deaccess file
IO.EXT	Extend file
IO.RAT	Read file attributes
IO.WAT	Write file attributes
IO.FNA	Find file name in directory
IO.RNA	Remove file name from directory
IO.ENA	Enter file name in directory
IO.ULK	Unlock block

MTAACP supports the following functions:

IO.FNA	Find file by name
IO.ENA	Enter name in directory (a no-op)
IO.ACR	Access for read only
IO.ACW	Access for read/write
IO.ACE	Access for read/write/extend
IO.DAC	Deaccess file
IO.RVB	Read virtual block
IO.WVB	Write virtual block
IO.EXT	Extend file
IO.CRE	Create file
IO.RAT	Read attributes
IO.APC	ACP control
IO.APV	Privileged ACP control

C.1 QIO$ PARAMETER LIST FORMAT

The device-independent part of a file processing QIO$ parameter list is identical to all other QIO$ parameter lists. The general QIO parameter list is described in detail in Section 1.6 of this manual. The file processor QIO$s require the following six additional words in the parameter lists:

Parameter Word 1	Address of a 3-word block containing the file identifier
Parameter Word 2	Address of the attribute list

Parameter Words 3 & 4	Size and extend control information
Parameter Word 5	Window size information and access control
Parameter word 6	Address of the file name block

NOTE

The Micro/RSX Executive treats File Identifier Blocks, filename blocks, and attribute list entries as read/write data. For this reason, they may not be used in read-only code segments or libraries.

C.1.1 File Identification Block

The File Identification Block is a 3-word block containing the file number and the file sequence number. The format of the File Identification Block is shown in Figure C-1.

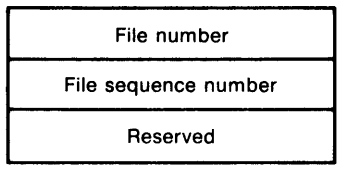


Figure C-1 File Identification Block

F11ACP uses the file number as an index to the file header in the index file. Each time a header block is used for a new file, the file sequence number is incremented. This insures that the file header is always unique. The third word is not currently used but is reserved for the future.

C.1.2 The Attribute List

The file attribute list controls F11ACP reads or writes. File attributes are fields in the file header. These fields are described in detail in the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual.

The attribute list contains a variable number of entries terminated by an all-0 byte. The maximum number of entries in the attribute list is six.

An entry in the attribute list has the following format:

```
.BYTE  <Attribute type>, Attribute size
.WORD  Pointer to the attribute buffer
```

C.1.2.1 **The Attribute Type** - This field identifies the individual attribute to be read or written. The sign of the attribute type code determines whether the transfer is a read or write operation. If the type code is negative, the ACP reads the attribute into the buffer.

If the type code is positive, the ACP writes the attribute to the file header. Note that the sign of the type code must agree with the direction implied by the operation. For example, if the type code is positive, the operation must be an IO.WAT or IO.DAC.

The attribute type is one of the following:

- **File owner (H.FOWN)**

  The file owner UIC is a binary word. The low byte is the owner number and the high byte is the group number.

- **File protection (H.FPRO)**

  The file protection word is a bit mask with the following format:

  Each of the fields contains four bits, as follows:

  ```
  Bit 1  Read Access
  Bit 2  Write Access
  Bit 3  Extend Access
  Bit 4  Delete Access
  ```

- **File characteristics (H.UCHA)**

  The following user characteristics are currently contained in the 1-byte H.UCHA field:

  ```
  UC.CON = 200   Logically contiguous file
  UC.DLK = 100   File improperly closed
  ```

- **Record I/O Area (U.UFAT)**

  This field contains a copy of the first seven words of the file descriptor block. (RMS uses 32 bytes. The first seven are compatible with FCS for sequential files.) See the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual for a description of the FDB.

- **File name (I.FNAM)**

  The file name is stored as nine Radix-50 characters. The fourth word of this block contains the file type and the fifth word contains the version number.

- **File type (I.FTYP)**

  The file type is stored as three Radix-50 characters.

- **Version number (I.FVER)**

  The version number is stored as a binary number.

- **Expiration date (I.EXDT)**

  Creation date (I.CRDT)
  Revision date (I.RVDT)

  The expiration date is currently unused. When the file is created, the ACP initializes the creation date to the current date and time. It initializes the expiration and revision dates to 0. The ACP sets the revision date to the current date and time each time the file is deaccessed.

- **Statistics block**

  This block is described in the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual.

- **Read entire file header**

  This buffer is assumed to be 1000 blocks long. You cannot write this attribute.

- **Revision number (I.RVNO)**

  The ACP sets the revision number to 0, and increments it every time the file is deaccessed.

- **Placement Control**

C.1.2.2 **Attribute Size** - This byte specifies the number of bytes of the attribute to be transferred. Legal values are from 1 to the maximum size of the particular attribute. Table C-1 shows the maximum size for each attribute type.

Table C-1
Maximum Size for Each File Attribute

Attribute Type Code	Attribute Type	Maximum Attribute Size in Octal Bytes
1	File owner	6
2	Protection	4
3	File characteristics	2
4	Record I/O area	40
5	File name,type,version number	12
6	File type	4
7	Version number	2
10	Expiration date	7
11	Statistics block	12
12	Entire file header	0
13	Block size (magtape only)	--
15	Revision number and creation/revision/expiration dates	43
16	Placement control	16

C.1.2.3 **Attribute Buffer Address** - The attribute buffer address field contains the address of the buffer in the user's task space to or from which the attribute is to be transferred.

C.1.3 Size and Extend Control

These two parameters specify how many blocks the file processor allocates to a new file or adds to an existing file. These parameters also control the type of block allocation.

The format is as follows:

```
.BYTE  <High 8 bits of size>, <extend control>
.WORD  <Low 16 bits of size>
```

The size field specifies the number of blocks to be allocated to a file on IO.CRE and IO.EXT operations, and the final file size on IO.DEL operations.

The extend control field controls the manner in which an extend operation is to be done. The following bits are defined:

EX.AC1=1	The extend size is to be added as a contiguous block.
EX.AC2=2	Extend by the largest available contiguous piece up to the specified size.
EX.FCO=4	The file must end up contiguous.
EX.ADF=10	Use the default rather than the specified size. The default extend size is the size that was specified when the volume was mounted.
EX.ALL=20	Placement control (see Section C.2).
EX.ENA=200	Enable extend.

C.1.4 Window Size and Access Control

This parameter specifies the window size and access control information in the following format:

```
.BYTE  <window size>, <access control>
```

This word is only processed if the high bit of the access control byte (AC.ENB) is set.

Window size is the number of mapping entries. Specifying a negative window size minimizes window turns. If this byte is zero, the file processor uses the volume default. The size of the window allocated in the dynamic storage region is 6 times the number of mapping entries (each mapping entry is 3 words), plus 10 bytes for the window control block.

On Micro/RSX systems with secondary pool support, the mapping entries are allocated in secondary pool. The window control block and a pointer to secondary pool are located in primary pool.

The following access control bits are defined:

AC.LCK=1	Lock out further accesses for Write or Extend
AC.DLK=2	Enable deaccess lock

The deaccess lock sets the lock bit in the file header if the file is deaccessed as the result of a task exit without explicitly deaccessing the file. The lock bit is set by the executive. The lock bit is not set when the system crashes.

AC.LKL=4	Enable block locking
AC.EXL=10	Enable explicit block unlocking
AC.ENB=200	Enable access
AC.RWD=10	Rewind the volume (labeled and unlabeled magtape only)
AC.UPD=100	Update mode (labeled magtape only)
AC.POS=20	Do not position to end-of-volume (labeled magtape only)
AC.WCK=40	Initiate driver write-checking

NOTE

Both AC.LKL and AC.EXL must be set if you want block locking. If you do not want block locking, both bits must be clear. Any other combination is an error.

C.1.5 File Name Block Pointer

This word contains the address of a 15-word block in the issuing task's space. This block is called the file name block. The file name block is described in detail in the RSX-11M/M-PLUS and Micro/RSX I/O Operations Reference Manual.

The fields of the file name block that are particularly important in file-processing operations are:

- **Directory identification (N.DID)**

  This field is required for all disk operations. It specifies the directory to which the operation applies. This field is not used for tape operations.

- **File identification (N.FID)**

  This field is required as input for enter operations. This field is returned as output by find and remove operations.

- **File name (N.FNAM), type (N.FTYP), and version number (N.FVER)**

  These fields are required as input to enter, find, and remove operations. For find and remove operations, the file processor locates the appropriate entry by matching the information in these fields with the directory entries.

- **Status word (N.STAT)**
- **Wildcard context (N.NEXT)**

  This field is required as input for wildcard operations. It specifies the point at which to resume processing. It is updated for the next operation. It must initially be set to 0.

C.2 PLACEMENT CONTROL

The placement control attribute list entry controls the placement of a file in a particular place on the disk. You can specify either exact or approximate placement on IO.CRE and IO.EXT operations.

The placement control entry must be the first entry in the attribute list.

The format of the placement control attribute list entry is as follows:

```
.BYTE          ; Placement control,0
.WORD          ; High-order bits of VBN or LBN
.WORK          ; Low-order bits of VBN or LBN
.BLKW  4       ; Buffer to receive starting and ending LBN if
                ; AL.LBN is set.
```

The following bits are defined for the placement control field:

AL.VBN=1	Set if block specified is a VBN; otherwise, the block is the LBN
AL.APX=2	Set if you want approximate placement; otherwise, placement is exact
AL.LBN=4	Set if you want starting and ending LBN information

C.3 BLOCK LOCKING

Block locking only occurs when the user accesses a file with AC.LKL and AC.EXL set in the access control byte of the parameter list. Any read or write operation causes a check to see if the block is locked.

A write access locks a block for exclusive access. A write operation can only access a block that is not locked by any accessor. The only exception to this is an exact match with a previous lock owned by the same accessor.

A read access locks a block for shared access. A read operation can access any block locked for shared access.

The user must unlock a block with an explicit unlock request, IO.ULK. IO.ULK may be used to unlock one or all blocks.

If all accessors to a file have not requested block locking, the ACP returns an error (see Table C-2).

When the file is deaccessed, all locks owned by the accessor are released.

Each active lock requires eight bytes from the dynamic storage region. This storage is deallocated when the file is deaccessed.

C.4 SUMMARY OF F11ACP FUNCTIONS

The following is a summary of the functions implemented in F11ACP. A list of accepted parameters follows each function. All parameters are required unless specified as optional. Parameters other than those listed are illegal for that function and must be 0.

IO.CRE **Create file**

#1	The file identifier block is filled in with the file identifier and sequence number of the created file.
#2	Write attribute list and placement control list or both (optional)
#3 & #4	Extend control (optional) The amount allocated to the file is returned in the high byte of IOST(1) plus IOST(2).
#5	May be nonzero but must be disabled

IO.DEL **Delete or truncate file**

#1	Optional if the file is accessed
#3 & #4	Size to truncate the file to. If not enabled, the file is deleted. If enabled, the remaining 31 bits specify the size the file is to be after truncation. The change in file allocation is returned in the high byte of IOST(1) plus IOST(2). This amount will be zero or negative.

IO.ACR **Access file for read only**

IO.ACW **Access file for read/write**

IO.ACE **Access file for read/write/extend**

#1	File identifier pointer
#2	Read attributes control (optional)
#5	Access control must be enabled

IO.DAC **Deaccess file**

#1	File identifier pointer (optional)
#2	Write attributes control list
#5	May be nonzero but must be disabled

IO.EXT **Extend file**

#1	Optional if file is accessed
#2	Placement control attribute list (optional)

	#3 & #4	Extend control
		The amount allocated to the file is returned in the high byte of IOST(1) plus IOST(2).
IO.RAT	**Read attributes**	
	#1	Optional if file is accessed
	#2	Read attributes control list
IO.FNA	**Find name in directory**	
IO.RNA	**Remove name from directory**	
IO.ENA	**Enter name in directory**	
	#5	May be nonzero but must be disabled
	#6	File name block pointer
IO.ULK	**Unlock block**	
	#2	0 or count of blocks to unlock
	#4 & #5	Starting VBN to unlock or 0 to unlock all blocks.
IO.RVB	**Read virtual block**	
IO.WVB	**Write virtual block**	
	#1	User buffer
	#2	Buffer length
	#4 & #5	VBN

C.5 SUMMARY OF MTAACP FUNCTIONS

The following is a summary of the functions implemented in MTAACP. A list of accepted parameters follows each function. All parameters are required unless specified as optional. Parameters other than those listed are illegal for that function and must be 0.

IO.FNA	**Find file by name**	
	#5	AC.RWD set in the access control byte indicates that the volume is to be rewound prior to the search.
	#6	Pointer to file name block. The following fields are used as input: N.FNAM N.FTYP N.FVER N.STAT

The following fields are returned by MTAACP:

N.FID
N.FNAM
N.FTYP
N.FVER
N.STAT

IO.ENA **Enter name in directory**

A no-op for magnetic tape

IO.ACR **Access for read only**

#1 File identifier pointer. Used to position a tape by file identifier.

#2 Read attribute list (optional)

#5 Ignored

IO.ACW **Access for read/write**

This function will be rejected with the error code IE.PRI. (Extend access is required.)

IO.ACE **Access for read/write/extend**

#1 File identifier pointer. Used to position tape by file identifier.

#2 Read attribute list (optional)

#5 AC.UPD (update mode). If AC.UPD is set, the tape will be positioned to overwrite the file and all files beyond the current file will be lost. If AC.UPD is not set, the tape will be positioned for append. If the file is not the last file, MTAACP returns the error code IE.ISQ.

IO.DAC **Deaccess file**

#1 File identifier pointer is ignored.

#5 AC.RWD set indicates that the volume is to be rewound after the file is closed.

IO.RVB **Read virtual block**

#1 Buffer address

#2 Buffer size. The buffer size must be greater than 18 bytes and less than the declared block length for the entire file.

#4 High VBN

#5 Low VBN

The virtual block number must be either zero or exactly one greater than the previous block number.

IO.CRE **Create File**

#1 File identifier pointer. The file sequence and section number will be returned to the user's file identifier block.

#2 Attribute list pointer. Used to write the attributes for the newly created file. Attribute type code must be positive.

#5 If AC.RWD is set, the volume will be positioned at the beginning and will overwrite the first file. This effectively reinitializes the volume.

If AC.RWD is not set and AC.POS is set, the volume set will be positioned to the next file position beyond the current file and will overwrite that file. All files beyond that on the volume will be destroyed.

If neither AC.RWD nor AC.POS is set, the volume set will be positioned at its end and the new file will be appended to the set.

For unlabeled tapes, MTAACP only checks AC.RWD.

#6 Filename block pointer.

IO.RAT **Read Attributes**

#1 File identifier pointer. Used to position the tape by the file identifier.

#2 Attribute list pointer (see Section C.1.2)

The following attribute list entries are meaningful for magnetic tape:

```
1,2   UIC
1,4   UIC and protection
1,5   UIC, protection, and
      characteristics
2,2   Protection
2.3   Protection and characteristics
3,1   Characteristics
4,32  User file attributes
5,6   File name
5,8   File name and type
5,10  File name and type
6,2   File type
6,4   File type and version number
7,2   Version number
8,7   Expiration date
-9,10 Statistics block (read only)
-10,0 Entire header (read only)
11,2  Block size
```

IO.APC ACP Control

#3 One of the following user control function codes:

1 Rewind volume set.
2 Position to end of volume set.
3 Close current volume and continue processing the next section of the same file on the next volume of the volume set.
4 Space physical records in currently accessed file.
5 Get ACP characteristics.
6 Rewind current file.

IO.APV Privileged ACP Control

This function is used only by the MOUNT and DISMOUNT commands. This interface is subject to change and, therefore, will not be documented until a future release.

C.6 HOW TO USE THE ACP QIO$ FUNCTIONS

Although the operations described in this appendix are normally performed by the file-access methods (RMS and FCS), your application may issue the ACP QIO$s. The required parameters for each QIO$ are described in the preceding section. The necessary steps for common operations are described in the following section:

NOTE

The file identifier is the only way to refer to a file.

C.6.1 Creating a File

To create a file:

- Use IO.CRE to create it.
- Enter it in the Master File Directory (MFD) or a user directory with IO.ENA.

C.6.2 Opening a File

To open a file:

- Use IO.FNA to find the File Identifier of the directory in the MFD.
- Use IO.FNA to find the File Identifier of the file in the directory.
- Access the file with IO.ACR, IO.ACW, or IO.ACE.

C.6.3 Closing a File

To close a file:

- Deaccess the file with IO.DAC.

C.6.4 Extending a File

To extend a file:

- Use IO.FNA to find the file identifier if the file is not accessed.
- Use IO.EXT to extend the file.

C.6.5 Deleting a File

To delete a file:

- Use IO.FNA to find the file identifier.
- Use IO.RNA to remove the directory name.
- Use IO.DEL to delete the file.

C.7 ERRORS RETURNED BY THE FILE PROCESSORS

The error codes returned by F11ACP and MTAACP are shown in Table C-2.

Table C-2
File Processor Error Codes

Error Code	Operations	Explanation
IE.ABO	IO.RVB/IO.WVB	Indicates that not all requested data was transferred by the device.
IE.ALC	Extend or create	Indicates that the operation failed to allocate the file because of placement control or because of other related problems.
IE.ALN	Access a file	Indicates that a file is already accessed on that LUN.
IE.BAD	Any function	Indicates that a required parameter is missing, that a parameter that should not be present is present, that a parameter that must be disabled is enabled, or that a parameter value is invalid.

(Continued on next page)

Table C-2 (Cont.)
File Processor Error Codes

Error Code	Operations	Explanation
IE.BDR	Directory	Indicates that you attempted a directory operation on a file that is not a directory, or that the specified directory is corrupted. This is usually caused by a 0 version number field.
IE.BHD	Any	Indicates that a corrupt file header was encountered, or that the operation required a feature not supported by the FCP (such as multiheader support or support for unimplemented features).
IE.BVR	Directory	Indicates that you attempted to enter a name in a directory with a negative or 0 version number.
IE.BYT	Any function	This error is returned if the buffer specified is on an odd byte boundary or is not a multiple of four bytes.
IE.BTP	Create unlabeled Magtape	An attempt was made to create an unlabeled tape file with a record type other than fixed.
IE.CKS	Any	Indicates that the checksum of a file header is incorrect.
IE.CLO	File access	Indicates that the file was locked against access by the "deaccess lock bit."
IE.DFU	Allocation request	Indicates that there is insufficient free disk space for the requested allocation.
IE.DUP	Enter name	Indicates that the name and version already exist.
IE.EOF	IO.RVB/IO.WVB/IO.DEL	On read operations, this indicates an attempt to read beyond end of file. On truncate operations, it indicates an attempt to truncate a file to a length longer than that allocated or that the file was already at EOF.

(Continued on next page)

Table C-2 (Cont.)
File Processor Error Codes

Error Code	Operations	Explanation
IE.HFU	Extended	Indicates that the file header is full and cannot contain any more retrieval pointers and that adding an extension header is not allowed. When this error code is returned on a create operation, it indicates that the index file could not be extended to allow a file header to be allocated.
IE.IFC	Executive	Illegal function code.
IE.IFU	Create or extend	Indicates that there are no file headers available based on the parameters specified when the volume was initialized.
IE.LCK	File access, directory, and truncate	Indicates that the file is already accessed by a writer and that shared write has not been requested or is not allowed.
IE.LUN	Any requiring file ID	Indicates that file ID has not been supplied and that the file is not accessed on the LUN.
IE.NOD	All requiring DSR	Indicates that an I/O request failed because of IE.UPN, and that the FCP was unable to allocate required space from DSR or from secondary pool for data structures.
IE.NSF	All file	Indicates that the specified directory entry does not exist, that a file corresponding to the file ID does not exist, or that the file is marked for delete.
IE.OFL	Executive	The device is off line.
IE.PRI	Any	Indicates that the user does not have the required privilege for the requested operation, or that the user has not requested the proper access to the file if the

(Continued on next page)

Table C-2 (Cont.)
File Processor Error Codes

Error Code	Operations	Explanation
		file is already accessed (for example, in an attempt to write to a file that is accessed for read). This error code also indicates an attempt to do file I/O to a device that is not mounted.
IE.RER	Any	Indicates that the FCP encountered a fatal device read error during an operation; the operation has been aborted.
IE.SNC	Any	Indicates that the file number and the value contained in the header do not agree. This generally means that the header has gone bad because of a crash or a hardware error.
IE.SPC	Executive	Indicates an illegal buffer.
IE.SQC	Any	Indicates that the file sequence number does not agree with the file header; usually indicates that the file has been deleted and the header has been reused.
IE.WAC	File access	Indicates that the file is already write accessed and lock against writers is requested.
IE.WAT	Write attributes and deaccess	Indicates that the FCP encountered an invalid attribute.
IE.WER	Any	Indicates that the FCP encountered a fatal device write error during an operation. The operation has been aborted, but the disk structure may have been corrupted.
IE.WLK	Any requiring write access	Indicates that the volume is software write-locked.

INDEX

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

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.

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

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

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

SSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03062-2698

Do Not Tear - Fold Here