Professional™

300 series

P/OS System Reference Manual

Order No. AD-N620A-T1

Developer's Tool Kit

digital

software

P/OS System Reference Manual

Order No. AD-N620A-T1

September 1983

This manual describes the Professional Operating System (P/OS). It allows system and application programmers to use the operating system resources to optimize the performance of applications written for the Professional.

DEVELOPMENT SYSTEM:	VAX/VMS V3.2 or later RSX-11M V4.1 or later RSX-11M-PLUS V2.1 or later P/OS V1.7
SOFTWARE:	Professional Host Tool Kit V1.7 PRO/Tool Kit V1.0

DIGITAL EQUIPMENT CORPORATION
Maynard, Massachusetts 01754

First Printing, December 1982
Updated, September 1983

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 only be used or copied 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 or its affiliated companies.

The specifications and drawings, herein, are the property of Digital Equipment Corporation and shall not be reproduced or copied or used in whole or in part as the basis for the manufacture or sale of items without written permission.

Copyright © 1982 by Digital Equipment Corporation
All Rights Reserved

The following are trademarks of Digital Equipment Corporation:

CTI BUS	MASSBUS	RSTS
DEC	PDP	RSX
DECmate	P/OS	Tool Kit
DECsystem-10	PRO/BASIC	UNIBUS
DECSYSTEM-20	Professional	VAX
DECUS	PRO/FMS	VMS
DECwriter	PRO/RMS	VT
DIBOL	PROSE	Work Processor
digital	Rainbow	

Update Notice Number 1

P/OS System Reference Manual
AD-N620A-T1

September 1983

Insert this page in the P/OS System Reference Manual to maintain an up-to-date record of changes to the manual.

NEW AND CHANGED INFORMATION

This update reflects software changes and additions made in P/OS Version 1.7. Also included are additional corrections to the documentation.

Copyright © 1983 Digital Equipment Corporation

INSTRUCTIONS

Add the following pages to the P/OS System Reference Manual as replacement for or additions to current pages. The technical changes made on replacement pages are indicated in the outside margin by change bars. Changes of an editorial nature are not marked. A date at the bottom of the new pages denotes revised or new information for this update.

OLD PAGE	NEW PAGE
Title page/copyright page	Title page/copyright page
iii through xi/blank	iii through xi/blank
8-1 through 8-16	8-1 through 8-26
9-25/9-26, 9/-27/9-28	9-25/9-26, 9/-27/9-28
9-149/9-150	9-149/9-150, 9-150.1/9-150.2
9-153/9-154	9-153/9-154
9-155/9-156	9-155/9-156
9-163/9-164, 9-165/166	9-163/9-164, 9-165/166
11-1 through 11-5/blank	11-1 through 11-7/blank
12-1 through 12-26	12-1 through 12-26
13-1 through 13-13/blank	13-1 through 13-12
C-1 through C-6	C-1 through C-6
Index I-1 through I-6	Index I-1 through I-6
Reader's Comments/Mailer	Reader's Comments/Mailer

CONTENTS

CHAPTER 8
CALLABLE SYSTEM ROUTINES

P/OS provides a set of callable routines in a resident library called POSSUM. The POSSUM routines and their functions are:

- ☐ PROATR—gets or sets a file's attributes (Section 8.2)
- ☐ PRODIR—creates or deletes a directory (Section 8.3)
- ☐ PROFBI—formats, initializes, and checks for bad blocks on a disk or diskette (Section 8.4)
- ☐ PROLOG—translates, creates, and deletes a logical name (Section 8.5)
- ☐ PROTSK—installs, removes, or fixes a task (Section 8.6)
- ☐ PROVOL—mounts, dismounts, bootstraps, and/or writes the bootblock on a volume (Section 8.7)

To use the routines, you must task build your program against the POSSUM resident library. A program calls a POSSUM routine by the PDP-11 standard R5 calling sequence (see Section 8.1.1). Some of the routines use a separate task in the system called a server. This chapter describes each callable routine as well as the name of any server that a particular routine may require.

POSSUM can be included as part of a cluster of libraries with RMSRES and other libraries. See the *RSX-11M/M-PLUS Task Builder Manual* for details on cluster libraries.

Note: When you link programs to run on the Professional, invoke the Task Builder using the name PAB (Professional Applications Builder) rather than TKB.

You can provide one of two options in your task build command file to include the POSSUM library in your task:

Use the following Task Builder format to link a task to the POSSUM resident library:

```
LIBR=POSSUM:RO
```

Use this Task Builder format to link a task to a cluster library which includes the POSSUM resident library:

```
CLSTR=POSSUM,OTHER:RO
```

8.1 GENERAL CONVENTIONS FOR ALL CALLABLE SYSTEM ROUTINES

This section defines the general mechanism used for calling all the defined system routines in the POSSUM resident library.

8.1.1 PDP-11 R5 Calling Sequence

Your program must use register 5 (R5) to pass the address of an argument list that resides in your task's data space. The argument list itself is of variable length, so that only the necessary arguments are passed.

The general MACRO-11 coding sequence of the call follows.

Instruction space coding sequence:

```
MOV     #ARGLST,R5      ; address of the argument list to pass

JSR     PC,SUB          ; call the subroutine
```

Data space coding sequence:

```
ARGLST: .BYTE   NUMBER,0        ; NUMBER is the number of arguments
                                ; following in the list

        .WORD   ADDR1           ; address of first argument

        ...                     ; other arguments

        .WORD   ADDRn           ; the nth argument
```

For higher level languages that support the R5 calling sequence (such as BASIC-PLUS-2 or FORTRAN-77), see your language reference manual or user guide for correct syntax. The examples in this chapter assume BASIC-PLUS-2 (BP2) as the high level language being used. All examples assume a higher level language call.

In BP2, you can invoke the previous MACRO-11 call as follows:

```
    120         CALL SUB BY REF (ADDR1%,...,ADDRn%)
```

BP2 internally formats an R5 calling block and issues the call to the system routine for you.

8.1.2 Conventions for Callable System Services

All of the routines documented in this chapter have specific conventions that you must follow:

- ☐ All arguments passed to the system routines are by reference. This means that you are passing the address of the value in your program to the routine in POSSUM.
- ☐ Every routine shares a common format in that the first argument (STATUS) is the address of an 8-word Status Control Block found in your program to which the routines return completion status. The Status Control Block is always eight words in length, so care must be taken to allocate the proper amount of space in your program.
- ☐ Every routine requires a REQUEST parameter. All of the routines are multipurpose and this 1-word REQUEST parameter is the method for specifying which option(s) to execute.
- ☐ When specifying either a device or file name string as a required element in an argument list, always specify the accompanying size field in bytes. (A byte corresponds to one ASCII character.)
- ☐ The system services preserve registers R0–R4. This is of no concern to higher level language programmers since the languages preserve internal registers around the call.

8.1.3 Status Control Block Format

The 8-word Status Control Block has the following format:

word 0	is the count of the number of status parameters passed back to the Status Control Block upon completion of the routine.
word 1	is the overall call status. This is a 1-word value defined as follows: +1 = Success –1 = Directive Status Error. The actual $DSW error is in word 2. –2 = A QIO error. The contents of the 2-word QIO status block are in words 2 and 3. –3 = An RMS-11 error. The RMS-11 STS and STV fields are returned in words 2 and 3. –4 = Server specific error. The contents of words 2 through 7 are defined for each routine in Sections 8.2 through 8.7.

-5 = Interface error. An error occured when trying to interpret the argument block. Currently, one of the following values would be in word 2:

- -1 = Feature not supported. The code is not yet complete to execute the documented feature.
- -2 = Impure area is invalid, or missing. Usually indicates that you have not correctly taskbuilt your program.
- -3 = Invalid number of parameters (too few or too many).
- -4 = Server not installed.
- -5 = Illegal device specification.
- -6 = User buffer too small for returned data.
- -7 = Incompatibility between POSSUM library and task. Relink task to resolve the incompatibility.

word 2 - 7 as defined above

Note: Words 2 through 7, depending on their use, may represent integers (e.g. error codes) or ASCII strings (e.g. volume labels in the case of the PROFBI and PROVOL routines).

The following sections describe each callable routine in detail.

8.2 PROATR

The PROATR routine provides two forms of accessing file attributes. You can use PROATR to:

- ☐ Get attributes of a file
- ☐ Set attributes of a file

Given a file ID and an attribute list, the GET function uses the attribute list to determine which attributes to read and where to store the associated information. Conversely, the SET function writes the attribute information specified in the attribute list to the file header.

The PROATR routine does not require a server to execute.

To get or set file attributes, invoke the PROATR routine with the following arguments:

STATUS, REQUEST, ATTRIBUTE_LIST, FILE_ID, LUN

where:

STATUS	The address of the 8-word Status Control Block
REQUEST	The address of a word containing the decimal value of the operation to be performed. The values are: 0 = Get file attributes 1 = Set file attributes

ATTRIBUTE_LIST	The address of the attribute list. The attribute list contains a variable number of two-word entries terminated by a byte containing the value 0. Each entry is associated with an accessible file attribute as defined in Table 8-1. The maximum number of entries in the attribute list is six.

An entry in the attribute list has the following format:

```
.BYTE   Attribute code
.BYTE   Size of attribute buffer
.WORD   Address of attribute buffer
```

The attribute code and the size of the attribute buffer (derived from Table 8-1) are in the low and high bytes, respectively, of the first word of each entry. For GET: The attribute buffer is initially empty and receives attribute information from the file header. For SET: The contents of the attribute buffer are transferred to the file header.

FILE_ID	The address of a word containing a 3-word Files-11 File ID (FID). The file identification block is a 3-word block containing the file number, the file sequence number, and a reserved word.

```
FID:   File number
+2:    File sequence number
+4:    Reserved
```

The values in the file identification block can be obtained from the FID field in the NAM block used by RMS-11.

LUN	The address of a buffer containing the LUN number used to obtain the file ID. The LUN number can be obtained from the LCH field in the FAB block used by RMS-11.

Table 8-1
Accessible File Attributes

Attribute Code (Octal)	*Attribute Type*	*Size in Octal Bytes of Attribute Buffer*
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
11	Statistics block	12
16	Placement control	16

Note: The file name contained in the header is not associated with the name in a directory entry except by convention. Therefore, you cannot use the file ID to get the file name as specified in the directory; the name that the ACP returns is the name contained in the header.

8.2.1 Status Codes Returned by PROATR

PROATR does not use a server and, therefore, does not return any server-specific status codes. For other status codes, refer to Section 8.1.3 (Status Control Block Format).

8.3 PRODIR

The PRODIR routine provides two forms of directory manipulation. You can use PRODIR to:

- ☐ Create a directory on a device
- ☐ Delete a directory on a device

The name of the server used to execute PRODIR is CREDEL. This server must be installed in your system to perform any of indicated services. Otherwise, PRODIR returns a directive error in the Status Control Block (see Section 8.1.3). To create or delete a directory, invoke the PRODIR routine with the following arguments:

STATUS, REQUEST, DIRECTORY_NAME, DIRECTORY_SIZE

where:

STATUS	The address of the 8-word Status Control Block
REQUEST	The address of a word containing the decimal value indicating the operation to be performed. The values are: 1 = Create directory 2 = Delete directory
DIRECTORY_NAME	The address of a buffer containing an ASCII device and directory specification The device specification takes the form ddn: where: dd = the device name n = the device unit number (For example, DZ1:) The directory specification takes one of the following forms: [ggg,mmm] such as [301,3] or [gggmmm] such as [301003] or [name] such as [WILEY]

	where:
	ggg = group
	mmm =member
DIRECTORY_SIZE	The address of a byte value containing the number of characters in FILE_NAME

The following BASIC-PLUS-2 example shows the use of PRODIR to create and delete directories.

```
10  ! Program to Create/Delete directories
    DIM Status%(7)                          ! Set up 8 word status array
    DIM Req$(2)
20  Req$(1) = "Create"
    Req$(2) = "Delete"
    PRINT "Create or Delete (C/D) :";
    LINPUT #0,Req$                         ! enter request operation
    Request% = 1                            ! set default request to create
    IF LEFT$(Req$,1) = "C" THEN GOTO 100    ! determine if request was default
    ELSE IF LEFT$(Req$,1) <> "D" THEN GOTO 20
    ELSE Request% = 2
100 PRINT "Name of Directory to "; Req$(Request%); " (ddn:[dirspec]) :";
    LINPUT #0,Dfile$                        ! enter dir name to create/delete
    CALL Prodir BY REF (Status%(), Request%, Dfile$, LEN(Dfile$))
    FOR K = 0 TO 7                          ! check status array after call
    PRINT "Status"; K, Status%(K)
    NEXT K
999 END
```

8.3.1 Status Codes Returned by PRODIR

PRODIR does not return any server-specific status codes. For other status codes, refer to Section 8.1.3 (Status Control Block Format).

8.4 PROFBI

The PROFBI routine provides the mechanism for preparing media for use on the system. The PROFBI routine allows you to:

- ☐ Format a volume
- ☐ Check a volume for bad blocks
- ☐ Initialize a volume

To format or initialize a volume or check it for bad blocks invoke the PROFBI routine with the following arguments:

STATUS, REQUEST, DEVICE_SPEC, DEVICE_SIZE, ATTRIBUTE_LIST, ATTRIBUTE_SIZE where:

STATUS	The address of the 8-word Status Control Block. When a volume is successfully initialized (STATUS = +1), the last six words of the Status Control Block contain the volume label expressed as ASCII code (1 byte/character).
REQUEST	The address of a word containing the decimal value indicating the operation to be performed. The values are:

1 = Format a volume (only works for the hard disk)

2 = Check a volume for bad blocks

4 = Initialize a volume

Note: When preparing the hard disk, specify the REQUEST code to either format or check for bad blocks, but not for both. Either code will perform both functions on RD50-type devices in the same operation.

DEVICE_SPEC	The address of a buffer containing a character string which is the device specification of the volume to be formatted, initialized, or checked for bad blocks (1 byte/character).
DEVICE_SIZE	The address of a word containing the number of characters in DEVICE_SPEC.
ATTRIBUTE_LIST	The address of the attribute list. The attribute list is a buffer of legal attributes (see Notes). Legal attributes in PROFBI are:

1 = Volume label

2 = ACS buffer (allocate checkpoint space)

ATTRIBUTE_SIZE	The address of a word containing the total size of the attribute list.

Note: The contents of the buffer for the ATTRIBUTE_LIST argument are optional. That is, you must specifiy the argument but the buffer need not contain a volume label or an ACS specification.

Notes:

1. The minimum length of DEVICE_SPEC is four characters—the three-character device mnemonic followed by a colon (such as DW1:). The device portion of DEVICE_SPEC must end with a colon.

   If you are initializing a volume, part of the device specification can be the volume label which may be up to 12 characters (in the form DW1:SPECTROSCOPY). You may also specify the volume label in the attribute list instead. If you specify the volume label in both the DEVICE_SPEC argument and the ATTRIBUTE_LIST argument, the DEVICE_SPEC argument overrides the ATTRIBUTE_LIST argument.

2. If you omit the volume label when initializing a volume, PROFBI creates a default volume label using the date and time the volume was initialized. The default volume label format is:

   DDMMMYYHHMMS

3. DEVICE_SPEC may also be a logical name string. The logical name string must end with a colon. The number of logical name translations cannot exceed eight. A ninth translation results in an error condition.

4. PROFBI requires the string supplied in the DEVICE_SPEC and DEVICE_SIZE arguments when initializing a volume or checking it for bad blocks. The DEVICE_SPEC argument is necessary when formatting a volume.

5. The ATTRIBUTE_LIST argument is the means of specifying optional parameters. The attribute list for PROFBI is simply a buffer of legal attributes. The high byte in the first word of the attribute list specifies the attribute type. The low byte specifies the size of the attribute list buffer in bytes.

   You can use the attribute list as an alternate way to specify a volume label. That is, you can omit the volume label in the DEVICE_SPEC argument and specify it in the ATTRIBUTE_LIST. However, if you specify the volume label in both arguments, PROFBI overrides the ATTRIBUTE_LIST specification with the label specified in DEVICE_SPEC.

6. The attribute list for PROFBI also contains two additional, contiguous words as the Allocate Checkpoint Space (ACS) buffer. The high byte in the first word of the ACS buffer (2) identifies it as the ACS buffer. The low byte in the buffer specifies the number of bytes in that buffer. The second word in the ACS block identifies the number of blocks in the checkpoint file.

7. You must badblock a volume before you can initialize it.

The following BASIC-PLUS-2 example shows the use of PROFBI to check a volume for bad blocks and then initialize the volume.

```
10  REM Sample program for testing PROFBI bad blocking and initialization
    MAP (Sarray) INTEGER Stat(7)             ! Setup 8 word status block
    MAP (Sarray) BYTE Volname(15)            ! Setup BYTE mapping of Status
    PRINT "Load floppy into DZ1. Press RESUME to continue."
    CALL Wtres BY REF()

70  CALL Profbi BY REF( Stat(), 2%, "DZ1:", 4%, "", 0%)

    GOSUB 1000                               ! print status returned

110  CALL Profbi BY REF( Stat(), 4%, "DZ1:APPLDATA", 12%, "", 0%)

     GOSUB 1000                              ! print status returned
     Volumelabel$ = ""
     FOR I% = 4% TO 15%                      ! translate ASCII volume label
       Volumelabel$ = Volumelabel$ + CHR$(Volname(I%))
     NEXT I%
     GOTO 9999

1000 REM     Subroutine to print status returned from PROFBI
       FOR I% = 0% TO 7%
         PRINT "Status"; I%; Stat(I%)
       NEXT I%
     RETURN

9999 END
```

8.4.1 Status Codes Returned by PROFBI

The server-specific status codes returned by PROFBI are listed in Table 8-2.

Section 8.3.1 describes the Status Control Block format.

- A success status code (+1) is returned in word 1 (second word) of the Status Control Block. In that case, for PROFBI, words 2 through 7 of the Status Control Block contain the volume label expressed as ASCII code.
- A server-specific error is indicated with the value -4 being returned in word 1 of the Status Control block. In addition, the particular error code value (see Table 8-2) is returned in word 2.
- The location of status codes from other sources is as specified in Section 8.1.3.

Table 8-2
PROFBI Status Codes (Server Specific)

Status Code	*Comment*
+1	SUCCESS
-1	ILLEGAL DEVICE
-2	DEVICE NOT IN SYSTEM
-3	FAILED TO ATTACH DEVICE
-4	BLOCK ZERO BAD—DISK UNUSABLE
-5	AT LEAST ONE LBN (0 THROUGH 25) IS BAD. CANNOT INITIALIZE—DISK UNUSABLE
-6	BAD BLOCK FILE OVERFLOW
-7	UNRECOVERABLE ERROR
-8.	DEVICE WRITE-LOCKED
-9.	DEVICE NOT READY
-10.	FAILED TO WRITE BAD BLOCK FILE
-11.	PRIVILEGE VIOLATION
-12.	DEVICE IS AN ALIGNMENT CARTRIDGE
-13.	FATAL HARDWARE ERROR
-14.	ALLOCATION FAILURE
-15.	I/O ERROR SIZING DEVICE
-16.	ALLOCATION FOR SYS FILE EXCEEDS VOLUME LIMIT
-17.	HOMEBLOCK ALLOCATE WRITE ERROR
-18.	BOOTBLOCK WRITE ERROR—DISK UNUSABLE
-19.	INDEX FILE BITMAP I/O ERROR
-20.	BAD BLOCK HEADER I/O ERROR
-21.	MFD FILE HEADER I/O ERROR
-22.	NULL FILE HEADER I/O ERROR
-23.	CHECKPOINT FILE HEADER I/O ERROR

Table 8-2 (Cont)
PROFBI Status Codes (Server Specific)

Status Code	*Comment*
-24.	MFD WRITE ERROR
-25.	STORAGE BITMAP FILE HEADER I/O ERROR
-26.	FAILED TO READ BAD BLOCK DESCRIPTOR FILE
-27.	VOLUME NAME TOO LONG
-28.	UNRECOGNIZED DISK TYPE
-29.	PREALLOCATION INSUFFICIENT TO FILL FIRST INDEX FILE HEADER
-30.	PREALLOCATED TOO MANY HEADERS FOR SINGLE HEADER INDEX FILE
-31.	PREALLOCATION INSUFFICIENT TO FILL FIRST AND SECOND INDEX FILE HEADERS
-32.	BAD BLOCK LIMIT EXCEEDED FOR DEVICE
-33.	DRIVER NOT RESIDENT
-34.	BITMAP TOO LARGE—INCREASE CLUSTER FACTOR
-35.	STORAGE BITMAP I/O ERROR
-36.	HOMEBLOCK I/O ERROR
-37.	INDEX FILE HEADER I/O ERROR
-38.	DISMOUNT OF DEVICE FAILED
-39.	CANNOT MOUNT DEVICE FOREIGN
-40.	CANNOT MOUNT DEVICE FILES-11
-41.	CANNOT FORMAT DZ—PREFORMATTED
-42.	CANNOT DETACH DEVICE
-43.	CHECKPOINT FILE HEADER OVERFLOW—SPECIFY SMALLER CHECKPOINT FILE
-44.	NON-ALPHANUMERIC CHARACTER(S) IN VOLUME NAME—ILLEGAL

8.5 PROLOG

The PROLOG routine provides five forms of logical name manipulation. You can use PROLOG as follows:

- ☐ Create a logical name for a device specification
- ☐ Delete a logical name for a device specification
- ☐ Translate a logical name to a device specification
- ☐ Set the default directory and/or device
- ☐ Show the default directory and device

The name of the server used to execute PROLOG is SUMLOG. This server must be installed in your system to perform any of indicated services. Otherwise, PROLOG returns a directive error in the Status Control Block (see Section 8.1.3)

Caution: Do not use logical or directory names with this routine that are used by the P/OS system. (see the *Tool Kit User's Guide*)

8.5.1 Creating or Translating a Logical Name

To create or translate a logical name, invoke the PROLOG routine with the following arguments:

STATUS, REQUEST, LOGICAL_NAME, LOGICAL_NAME_SIZE, EQUIVALENCE, EQUIVALENCE_SIZE

where:

STATUS	The address of the 8-word Status Control block
REQUEST	The address of a word containing the decimal value indicating the operation to be performed. The values are: 3 = Create logical; 4 = Translate logical
LOGICAL_NAME	The address of a buffer containing an ASCII string (which can contain alphanumeric characters only, 1 byte/character). Refer to the *Tool Kit User's Guide* for P/OS conventions regarding logical names, device names, and reserved names.
LOGICAL_NAME_SIZE	The address of a byte value containing the number of characters in LOGICAL_NAME
EQUIVALENCE	The address of a buffer containing an ASCII device specification (1 byte/character). The device specification takes the form ddn: where: dd the device name; n the device unit number (for example, DZ1:)
EQUIVALENCE_SIZE	For CREATE: The address of a byte value containing the number of characters in EQUIVALENCE. For TRANSLATE: The address of a byte value containing the maximum number of characters in the EQUIVALENCE buffer.

For the TRANSLATE function, the EQUIVALENCE argument is an output argument returned by PROLOG. The length of the string returned in the EQUIVALENCE buffer is returned in the third word of STATUS.

The following BASIC-PLUS-2 example shows the use of PROLOG to create and then translate a logical name.

```
10  REM   Sample program to create and then translate a logical

    DIM Stat%(7%)                             ! set up status array
    Req% = 3%                                 ! set request to create
    Logicalname$ = 'Applicdisk'               ! logical name for create
    Logicalsize% = LEN(Logicalname$)          ! length of logical name
    Device$ = 'DZ1:'                          ! device to create logical on
    Length% = LEN(Device$)                    ! length of device specification

90  CALL Prolog BY REF(Stat%(), Req%,Logicalname$,Logicalsize%,Device$,Length%)

    GOSUB 400                                 ! print returned status
    Req% = 4%                                 ! set request to translate
    Eqv$ = SPACE$(40%)                        ! buffer for translated logical

140 CALL Prolog BY REF(Stat%(), Req%, Logicalname$,Logicalsize%,Eqv$,LEN(Eqv$))

    GOSUB 400                                 ! print returned status
    Eqv$ = TRM$(Eqv$)                         ! get rid of spaces from buffer
    PRINT 'Translated logical = '; Eqv$       ! print translated logical
    GOTO 999

400 REM      Subroutine to print status returned from PROLOG calls
    FOR I% = 0% TO 7%
      PRINT 'Status'; I%; ' '; Stat%(I%)
    NEXT I%
    RETURN
999 END
```

8.5.2 Deleting a Logical name and Set/Show

To delete a logical name or to set or show the default device and/or directory, invoke the PROLOG routine with the following arguments:

STATUS, REQUEST, LOGICAL_NAME, LOGICAL_NAME_SIZE,

where:

STATUS	The address of the 8-word Status Control block
REQUEST	The address of a word containing the decimal value indicating the operation to be performed. The values are: 1 = Set default 2 = Show default 5 = Delete logical

LOGICAL_NAME	The address of a buffer containing an ASCII string (1 byte/character) which can contain alphanumeric characters only. The user must have already created the LOGICAL_NAME.
LOGICAL_NAME_SIZE	For SET and DELETE: The address of a byte value containing the number of characters in LOGICAL_NAME. For SHOW: The address of a byte value containing the maximum number of characters in the LOGICAL_NAME buffer.

For the SET DEFAULT function, the LOGICAL_NAME string may contain a directory specification of the form

```
USERDISK:[DIRECTORY]
```

Where USERDISK: is the logical name with the directory specification appended to it.

The directory specification takes one of the following forms:

[ggg,mmm]	such as [301,3]
	or
[gggmmm]	such as [301003]
	or
[name]	such as [WILEY]
where:	
ggg	group
mmm	number

Note: When issuing a call for the SET DEFAULT function, note that the user can specify either the logical name or the directory. If you specify both, then both the default device and directory are changed. If you specify only one, the other does not change.

For both the DELETE and SET DEFAULT functions, there is no output argument; PROLOG returns the call status in the Status Control Block. For SET DEFAULT, PROLOG does not check whether the device or directory does in fact exist. No error status code is returned if the device or directory does not exist. PROLOG uses the SDIR$ directive, which also does not check whether the directory exists.

For the SHOW DEFAULT function, LOGICAL_NAME is an output argument returned by PROLOG. The LOGICAL_NAME also contains the default directory string. The length of the string returned in LOGICAL_NAME is returned in the third word of STATUS.

The following BASIC-PLUS-2 example shows the use of PROLOG to delete a logical, show the default directory, and set a current directory.

```
10  REM Sample program to delete a logical, show default directory, and set
    REM        current directory.
    DIM Stat%(7%)                             ! set up status array
    Logicalname$ = 'Applicdisk'               ! delete an existing logical
    Req% = 5%                                 ! set request to delete

200 CALL Prolog BY REF(Stat%(), Req%, Logicalname$, LEN(Logicalname$))

    GOSUB 400                                 ! print returned status
    Req% = 2%                                 ! set request to show default
    Showdir$ = SPACE$(40%)                    ! buffer for default directory
    Length% = LEN(Showdir$)                   ! length of buffer

250 CALL Prolog BY REF(Stat%(), Req%, Showdir$, Length%)

    GOSUB 400                                   ! print returned status
    PRINT 'Default directory = ';TRM$(Showdir$) ! get rid of spaces from buffer
    PRINT 'Directory name length = '; Stat%(2%) ! dir length stored in Stat(2%)
    Req% = 1%                                   ! set request to set default
    Dirname$ = 'USERDISK:[USERFILES]'           ! default directory

320 CALL Prolog BY REF(Stat%(), Req%, Dirname$, LEN(Dirname$))

    GOSUB 400                                   ! print status returned
    GOTO 999

400 REM      Subroutine to print status returned from PROLOG
    FOR I% = 0% TO 7%
      PRINT 'Status'; I%; ' '; Stat%(I%)
    NEXT I%
    RETURN
999 END
```

8.5.3 Status Codes Returned by PROLOG

Most error returns from PROLOG are Directive Status errors (see CLOG$, DLOG$, and TLOG$ logical name directives in Chapter 9. The numerical equivalents of the status codes are in the appendices).

The server-specific status codes returned by PROLOG are listed in Table 8-3.

Section 8.3.1 describes the Status Control Block format.

- ☐ A success status code (+1) is returned in word 1 (second word) of the Status Control Block.
- ☐ A server-specific error is indicated with the value -4 being returned in word 1 of the Status Control block. In addition, the particular error code value (see Table 8-3) is returned in word 2.
- ☐ The location of status codes from other sources is as specified in Section 8.1.3.

Table 8-3
PROLOG Status Codes (Server Specific)

Status Code	*Comment*
+1	SUCCESS
-1	ERROR IN PARSING THE SET DEFAULT STRING. EITHER A BADLY FORMED SPECIFICATION WAS PASSED OR SOMETHING OTHER THAN A DEVICE OR DIRECTORY WAS FOUND IN STRING.
-2	CANNOT DETERMINE TYPE OF SERVICE REQUESTED.

8.6 PROTSK

The PROTSK routine provides four forms of task manipulation. You can use PROTSK to:

- ☐ Install a task, region, or common
- ☐ Remove a task, region, or common
- ☐ Fix an installed task, region, or common in memory
- ☐ Install, run, and remove an offspring task through a parent task

In addition, you can specify that an installed task not be aborted or removed if the application exits or if the user presses INTERRUPT/DO at the terminal. This feature, called NOREMOVE, may be used, for example, to ensure that a non interactive "background" task, such as a file transfer, is not aborted inadvertently.

> Caution: Use care with the NOREMOVE option. If the name of the task is the same as that used by another application which is to be run subsequently, the second task of that name will not be installed unless the first is removed. Also, unless the install/run/remove option is specified, the only means of removing a task installed with the NOREMOVE option is either by powering down the system or by running an application that, knowing the task's name, can remove it. For example you can remove the task with the DCL REMOVE command on the PRO/Tool Kit (Tool Kit running on the Professional). See the descriptions of the REQUEST argument for Install a Task and for Install/Run/Remove a Task.

The name of the server used to execute PROTSK is INSREM. This server must be installed in your system to perform any of indicated services. Otherwise, PROTSK returns a directive error in the Status Control Block (see Section 8.1.3)

8.6.1 Install a Task

To install a task, call the routine PROTSK with all the following arguments:

```
STATUS, REQUEST, TASK_NAME, FILE_NAME, FILE_SIZE
```

where:

STATUS	The address of the 8-word Status Control block
REQUEST	The address of a word containing the decimal value indicating the type of task manipulation to be performed: 1 = Install a task 4 = Fix a task 8 = Common, library, or region is read only 32 = Install task with name supplied in the TASK_NAME argument 64 = NO REMOVE. The task will not be aborted or removed if the application exits or if the user presses INTERRUPT/DO. In addition the task will not be removed upon exit. (see CAUTION at beginning of PROTSK section). If you require more than one task manipulation function, add the decimal values of each function together to produce a single decimal value. For example, to install a task and fix it in memory with your Radix-50 task name, specify the REQUEST value as 37. Obtain the REQUEST value by adding the values for install (1), fix (4), and name supplied in the TASK_NAME argument (32).
TASK_NAME	The address of a 2-word Radix-50 task name. Upon completion, PROTSK returns the 2-word Radix-50 installed task name at this location. If you selected value 32 as the REQUEST option (install task with name supplied in TASK_NAME), then supply the Radix-50 task name in the two words at the address before calling PROTSK.
FILE_NAME	The address of the buffer containing an ASCII file specification of the task image to be installed.
FILE_SIZE	The address of value describing the number of characters in FILE_NAME.

Caution: If you want to checkpoint a task, you must use the /CP Task Builder switch when building that task. In addition, you must have previously created a checkpoint file on your system disk.

The following BASIC-PLUS-2 example shows the use of PROTSK to install a task.

```
10  DIM STATUS%(7),TASKN%(1)             !Set up STATUS block (array)
    TYPE% = 1                            !REQUEST value (install)
    TFILE$="DZ1:[MYTASKS]PAYROLL1.TSK"   !FILE_NAME
    CALL PROTSK BY REF (STATUS%(),TYPE%, TASKN%(),TFILE$,LEN(TFILE$))
    GOTO 100 IF STATUS%(1) = 1           !Check for proper install
    PRINT "Error -- task did not install properly" \ STOP
100 TOTAL% = 0
      .
      .
      .
999 END
```

8.6.2 Remove a Task, Region, or Common

To remove a task, region, or common, call the routine PROTSK with all of the following arguments:

STATUS, REQUEST, TASK_NAME

where:

STATUS	The address of the 8-word Status Control block
REQUEST	indicating the type of task manipulation to be performed: 2 = Remove a task 16 = Task is a region or a common If you require more than one task manipulation function, add the decimal values of each function together to produce a single decimal value.
TASK_NAME	The address of a 2-word Radix-50 task name that identifies the task, region, or common to be removed.

8.6.3 Fix a Task, Region, or Common in Memory

To fix a task, region, or common in memory, call the routine PROTSK with the following arguments:

STATUS, REQUEST, TASK_NAME

where:

STATUS	The address of the 8-word Status Control Block

REQUEST	The address of a word containing the decimal value indicating the type of task manipulation to be performed: 4 = Fix a task in memory 16 = Task is a region or common If you require more than one task manipulation function, add the decimal values of each function together to produce a single decimal value.
TASK_NAME	The address of a 2-word Radix-50 task name that identifies the task, region, or common to be fixed in memory

8.6.4 Install/Run/Remove an Offspring Task

By requesting install/run/remove, a parent task can install an offspring task, have it run immediately, and have it removed upon exit. Install/run/remove can be executed in two ways, which may be compared to either "calling" the offspring task or "chaining" to the offspring task. The distinction between "Call" and "Chain" install/run/remove is as follows:

- With "Call", the parent task initiates execution of the offspring task and still continues its own execution. While the offspring is being installed and started, the parent is stopped. After the install/run/remove request has been completed, the parent is able to continue its own execution.
- With "Chain", the parent task initiates execution of the offspring task, passes offspring information, and then exits. If the offspring has been installed and initiated successfully, the parent exits. Otherwise, an error condition is returned to the parent, and the parent does not exit. The parent must perform any necessary cleanups, such as closing files, before chaining to the offspring.

To use install/run/remove, call the routine PROTSK with the following arguments:

```
STATUS, REQUEST, TASK_NAME, FILE_NAME, FILE_SIZE
[,COMMAND_LINE, COMMAND_SIZE] [,EVENT_FLAG]
[,EXIT_STATUS]
```

The arguments are defined below. Use all of the arguments that are not enclosed in brackets. The arguments enclosed in brackets are optional, with the following provisions:

- If none of the optional arguments are used, they may all be omitted.
- EVENT_FLAG and EXIT_STATUS are used only with the "Call" install/run/remove option.
- The optional arguments are positional. If EVENT_FLAG and/or EXIT_STATUS is used but COMMAND_LINE is not used, then the word in COMMAND_SIZE must have the value 0.

The arguments are defined as follows:

STATUS	The address of the 8-word Status Control block
REQUEST	indicating the type of task manipulation to be performed:

3 =	Install/run/remove a task
32 =	Install task with name supplied in the TASK_NAME argument
64 =	NO REMOVE. The task will not be aborted or removed if the application exits or the user presses INTERRUPT/DO. However, the task will be removed upon exit (see CAUTION at beginning of PROTSK section).
128 =	To select the "Call" install/run/remove option, add 128. to the decimal value in the REQUEST word. To select "Chain", diregard this value ("Chain" is the default option).

If you require more than one task manipulation function, add the decimal values of each function together to produce a single decimal value.

For example, to select install/run/remove with the "Call" option and your Radix-50 task name, specify the REQUEST value as 163. Obtain the REQUEST value by adding the values for install/run/remove (3), name supplied in the TASK_NAME argument (32), and the "Call" option (128).

When specifying install/run/remove, it is illegal to also specify "Fix a task in memory" or "Install a common or library".

TASK_NAME	The address of a 2-word Radix-50 (offspring) task name. Upon completion, PROTSK returns the 2-word Radix-50 installed task name at this location. If you selected value 32 as the REQUEST option (install task with name supplied in TASK_NAME), then supply the Radix-50 task name in the two words at the address before calling PROTSK.
FILE_NAME	The address of the buffer containing an ASCII file specification of the (offspring) task image to be installed.
FILE_SIZE	The address of value describing the number of characters in FILE_NAME.
COMMAND_LINE	(Optional). The address of a buffer containing a command line to be queued to the offspring task.

COMMAND_SIZE	The address of a value describing the number of characters in COMMAND_LINE. The maximum number of characters is 255 (decimal). If COMMAND_LINE is not specified, then the word in COMMAND_SIZE must have the value 0.
EVENT_FLAG	(For "Call" option only). The event flag to be cleared on issuance and set when the offspring task exits or emits status.
EXIT_STATUS	(For "Call" option only). The address of an 8-word status block to be written when the offspring task exits or emits status.

Word 0	=	Offspring task exit status
Word 1	=	System abort code
Word 2–7	=	Reserved

Note: The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical OR of the symbol SP.WX8 and the event flag number in the EVENT_FLAG parameter above.

Caution: If you want to checkpoint a task, you must use the /CP Task Builder switch when building that task. In addition, you must have previously created a checkpoint file on your system disk.

8.6.5 Status Codes Returned by PROTSK

The server-specific status codes returned by PROTSK are listed in Table 8-4.

Section 8.3.1 describes the Status Control Block format.

- ☐ A success status code (+1) is returned in word 1 (second word) of the Status Control Block.
- ☐ A server-specific error is indicated with the value -4 being returned in word 1 of the Status Control block. In addition, the particular error code value (see Table 8-4) is returned in word 2.
- ☐ The location of status codes from other sources is as specified in Section 8.1.3.

Table 8–4
PROTSK Status Codes (Server Specific)

Status Code	*Comment*
+1	SUCCESS
-1	TASK NAME IN USE
-2	FILE NOT FOUND
-3	SPECIFIED PARTITION TOO SMALL
-4	TASK AND PARTITION BASE MISMATCH
-7	LENGTH MISMATCH COMMON BLOCK
-8.	BASE MISMATCH COMMON BLOCK

Table 8-4 (Cont.)

Status Code	*Comment*
-9.	TOO MANY COMMON BLOCK REQUESTS
-11.	CHECKPOINT AREA TOO SMALL
-13.	NOT ENOUGH APRS FOR TASK IMAGE
-14.	FILE NOT A TASK IMAGE
-15.	BASE ADDRESS MUST BE ON 4K BOUNDARY
-16.	ILLEGAL FIRST APR
-18.	COMMON BLOCK PARAMETER MISMATCH
-20.	COMMON BLOCK NOT LOADED
-22.	TASK IMAGE VIRTUAL ADDRESS OVERLAPS COMMON BLOCK
-23.	TASK IMAGE ALREADY INSTALLED
-24.	ADDRESS EXTENSIONS NOT SUPPORTED
-26.	CHECKPOINT SPACE TOO SMALL, USING CHECKPOINT FILE
-27.	NO CHECKPOINT SPACE, ASSUMING NOT CHECKPOINTABLE
-29.	ILLEGAL UIC
-30.	NO POOL SPACE
-31.	ILLEGAL USE OF PARTITION OR REGION
-32.	ACCESS TO COMMON BLOCK DENIED
-33.	TASK IMAGE I/O ERROR
-34.	TOO MANY LUNS
-35.	ILLEGAL DEVICE
-36.	TASK MAY NOT BE RUN
-37.	TASK ACTIVE
-39.	TASK FIXED
-40.	TASK BEING FIXED
-41.	PARTITION BUSY
-43.	COMMON/TASK NOT IN SYSTEM
-44.	REGION OR COMMON FIXED
-45.	CANNOT DO RECEIVE FROM REQUESTOR
-46.	CANNOT ATTACH TO REQUESTOR
-47.	INVALID REQUEST
-48.	CANNOT RETURN STATUS
-49.	ERROR ENCOUNTERED ON FILE OPEN OPERATION
-50.	ERROR ENCOUNTERED ON FILE CLOSE OPERATION
-51.	CANNOT GET FILE LBN TO PROCESS LABEL BLOCKS

8.7 PROVOL

The PROVOL routine provides a twofold service. You can use PROVOL to:

- ☐ Mount or dismount a disk volume
- ☐ Write a bootblock on a volume and/or bootstrap a volume

To mount or dismount a volume, write a bootblock on a volume, or bootstrap a volume, invoke the PROVOL routine with the following arguments:

STATUS, REQUEST, DEVICE_SPEC, DEVICE_SIZE, ATTRIBUTE_LIST, ATTRIBUTE_SIZE

where:

STATUS	The address of the 8-word Status Control Block. When mounting a non-foreign volume, the last six words of the Status Control Block contain the volume label expressed as ASCII code (1 byte/character), provided that the operation is successful (STATUS = +1).
REQUEST	The address of a word containing the decimal value indicating the operation to be performed. The values are: 0 = Mount a volume 1 = Mount a foreign volume 2 = Dismount a volume 8 = Bootstrap a volume 9 = Write a bootblock on a volume 10 = Write a bootblock on a volume and bootstrap it
DEVICE_SPEC	The address of a buffer containing a character string (1 byte/character) which is the device specification of the volume to be mounted, dismounted, bootstrapped, or on which a bootblock is to be written.
DEVICE_SIZE	The address of a word containing the number of characters in DEVICE_SPEC.
ATTRIBUTE_LIST	The address of the attribute list. The attribute list is a buffer of legal attributes (see Notes). Legal attributes in PROVOL are: 1 = Volume label
ATTRIBUTE_SIZE	The address of a word containing the size of the attribute list.

Note: The contents of the buffer for the ATTRIBUTE_LIST argument are optional. That is, you must specifiy the argument but the buffer need not contain a volume label.

Notes:

1. The minimum length of DEVICE_SPEC is four characters—the three-character device mnemonic followed by a colon (such as DW1:). The device portion of DEVICE_SPEC must end with a colon.

   Part of the device specification can be the volume label which may be up to 12 characters. If you omit the volume label from DEVICE_SPEC, PROVOL gets the label from the specified disk by default. Whenever you specify a volume label in a DEVICE_SPEC argument (when mounting a volume, for example), the specified label must match the label on the volume; otherwise, the operation fails.

2. DEVICE_SPEC may also be a logical name string. In this case, the logical name string must end with a colon. The number of logical name translations cannot exceed eight. A ninth translation results in an error condition.

3. PROVOL requires the string supplied in the DEVICE_SPEC and DEVICE_SIZE arguments when mounting or dismounting a volume. The specified volume label must match the label on the volume for the operation to be successful. PROVOL ignores the volume label if mounting or dismounting a "foreign" volume.

4. PROVOL uses the string supplied in the DEVICE_SPEC and DEVICE_SIZE arguments to bootstrap a volume. The DEVICE_SPEC string may be a logical name. The number of logical name translations cannot exceed eight. A ninth translation results in an error condition.

5. When writing a bootblock to a volume, PROVOL requires a complete device, directory and file name specification. If you omit the file name, PROVOL uses the default directory and file name of [1,54]RSX11M.SYS.

6. The ATTRIBUTE_LIST argument is the means of specifying optional parameters. The attribute list for PROVOL is simply a buffer of legal attributes. The high byte in the first word of the attribute list specifies the attribute type. The low byte specifies the size of the buffer in bytes.

   You can use the attribute list as an alternate way to specify a volume label. That is, you can omit the volume label in the DEVICE_SPEC argument and supply it in the ATTRIBUTE_LIST argument. However, if you specify the volume label in both arguments, PROVOL overrides the ATTRIBUTE_LIST specification with the label specified in DEVICE_SPEC.

The following BASIC-PLUS-2 example shows the use of PROVOL to mount and dismount a volume.

```
10  REM Sample program to test PROVOL requests.
    MAP (Sarray) INTEGER Stat(7)
    MAP (Sarray) BYTE Volname(15)
    Device$ = 'DZ1:'                    ! setup device_spec parameter
    Devlen% = LEN(Device$)              ! setup device_size
    PRINT 'Insert floppy into DZ1:. Press RESUME to continue.'
    CALL Wtres BY REF()
    Req% = 2%                           ! dismount request since closing door
                                        ! causes mount to automatically happen
76  CALL Provol BY REF(Stat(), Req%, Device$, Devlen%, '',0%)

    GOSUB 2000                          ! print return status from dismount
    Req% = 1%                           ! mount foreign request

90  CALL Provol BY REF(Stat(), Req%, Device$, Devlen%, '',0%)

    GOSUB 2000                       ! print returned status from mount foreign
    .
    .
    .
    .
     Req% = 2%                          ! dismount request
     Device$ = "DZ1:"                   ! setup device_spec parameter
     Devlen% = LEN(Device$)             ! setup device_size parameter

230  CALL Provol BY REF(Stat(), Req%, Device$, Devlen%, '',0%)

     GOSUB 2000                         ! print returned status from dismount
     Req% = 0%                          ! mount request

290  CALL Provol BY REF(Stat(), Req%, Device$, Devlen%, '',0%)

     GOSUB 2000                         ! print returned status from mount
     Volumelabel$ = ""
     FOR I% = 4% TO 15%                 ! translate ASCII volume label in Stat
       Volumelabel$ = Volumelabel$ + CHR$(Volname(I%))
     NEXT I%
       .
       .
       .
     Req% = 9%                          ! write bootblock request
     Device$ = "DZ1:[ZZSYS]BOOT.SYS"    ! setup device-spec NOTE: This file
                                        ! must already exist. The volume
     Devlen% = LEN(Device$)             ! setup device_size

420  CALL Provol BY REF(Stat(), Req%, Device$, Devlen%, '', 0%)

     GOSUB 2000                         ! print returned status
     IF Stat(1) <> 1% THEN GOTO 9999    ! If write bootblock not successful end
     Req% = 8%                          ! bootstrap volume request
     Device$ = "DZ1:"                   ! setup device_spec
     Devlen% = LEN(Device$)             ! setup device_size

490  CALL Provol BY REF(Stat(), Req%, Device$, Devlen%, '', 0%)

     GOTO 9999

2000 REM   Subroutine for printing status returned from PROVOL calls
     FOR I% = 0% TO 7%
       PRINT 'Status'; I%; ' '; Stat(I%)
     NEXT I%
     RETURN
9999 END
```

8.7.1 Status Codes Returned by PROVOL

The server-specific status codes returned by PROVOL are listed in Table 8-5.

Section 8.3.1 describes the Status Control Block format.

- ☐ A success status code (+1) is returned in word 1 (second word) of the Status Control Block. In that case, for PROVOL, words 2 through 7 of the Status Control Block contain the volume label expressed as ASCII code.
- ☐ A server-specific error is indicated with the value -4 being returned in word 1 of the Status Control block. In addition, the particular error code value (see Table 8-5) is returned in word 2.
- ☐ The location of status codes from other sources is as specified in Section 8.1.3.

Table 8-5
PROVOL Status Codes (Server Specific)

Status Code	*Comment*
+1	SUCCESS
-1	FILE IS NOT A SYSTEM IMAGE
-2	INVALID BOOT DEVICE

9.1.11 CRRG$—Create Region

The Create Region directive creates a dynamic region in a system-controlled partition and optionally attaches it to the issuing task.

If RS.ATT is set in the region status word, the Executive attempts to attach the task to the newly created region. If no region name has been specified, the user's program must set RS.ATT (see the description of the Attach Region directive).

By default, the Executive marks a dynamically created region for deletion when the last task detaches from it. To override this default condition, set RS.NDL in the region status word as an input parameter. Be careful in considering to override the delete-on-last-detach option. An error within a program can cause the system to lock by leaving no free space in a system-controlled partition.

If the region is not given a name, the Executive ignores the state of RS.NDL. All unnamed regions are deleted when the last task detaches from them.

Named regions are put in the Common Block Directory (CBD). However, memory is not allocated until the Executive maps a task to the region.

The Executive returns an error if there is not enough space to accommodate the region in the specified partition. (See Notes.)

Fortran Call

```
CALL CRRG  (irdb[,ids])
```

irdb	An 8-word integer array containing a Region Definition Block (see Section 7.5.1.2)
ids	Directive status

Macro Call

```
CRRG$ rdb
```

rdb	Region Definition Block address

Macro Expansion

```
CRRG$    RDBADR
.BYTE    55.,2         ;CRRG$ MACRO DIC, DPB SIZE = 2 WORDS
.WORD    RDBADR        ;RDB ADDRESS
```

Table 9-3
Region Definition Block Parameters

Input Parameters

Array Element	*Offset*	*Description*	
irdb(2)	R.GSIZ	Size, in 32-word blocks, of the region to be created	
irdb(3)(4)	R.GNAM	Name of the region to be created, or 0 for no name	
irdb(5)(6)	R.GPAR	Name of the system-controlled partition in which the region is to be allocated, or 0 for the partition in which the task is running	
irdb(7)	R.GSTS	Bit settings* in the region status word:	
		Bit	*Definition (if bit=1)*
		RS.CRR	Region was successfully created.
		RS.UNM	At least one window was unmapped on a detach.
		RS.MDL	Mark region for deletion on last detach.
		RS.NDL	The region should not be deleted on last detach.
		RS.ATT	Created region should be attached.
		RS.NEX	Created region is not extendible.
		RS.RED	Read access is desired on attach.
		RS.WRT	Write access is desired on attach.
		RS.EXT	Extend access is desired on attach.
		RS.DEL	Delete access is desired on attach.
irdb(8)	R.GPRO	Protection word for the region (DEWR,DEWR,DEWR,DEWR)	

Output Parameters

irdb(1)	R.GID	ID assigned to the created region (returned if RS.ATT=1)	
irdb(2)	R.GSIZ	Size in 32-word blocks of the attached region (returned if RS.ATT=1)	
irdb(7)	R.GSTS	Bit settings* in the region status word:	
		Bit	*Definition*
		RS.CRR	1 if the region was successfully created

* If you are a Fortran programmer, refer to Section 7.5.1 to determine the bit values represented by the symbolic names described.

Local Symbol Definitions

C.RRBA	Region Definition Block address (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.UPN	A Partition Control Block (PCB) or an attachment descriptor could not be allocated, or the partition was not large enough to accommodate the region, or there is currently not enough continuous space in the partition to accommodate the region.
IE.HWR	The directive failed in the attachment stage because a region parity error was detected.
IE.PRI	Attach failed because desired access was not allowed.
IE.PNS	Specified partition in which the region was to be allocated does not exist; or no region name was specified and RS.ATT = 0.
IE.ADP	Part of the DPB or RDB is out of issuing task's address space.
IE.SDP	DIC or RDB size is invalid.

Notes

1. The Executive does not return an error if the named region already exists. In this case, the Executive clears the RS.CRR bit in the status word R.GSTS. If RS.ATT has been set, the Executive attempts to attach the already existing named region to the issuing task.
2. The protection word (see R.GPRO above) has the same format as that of the file system protection word. There are four categories, and the access for each category is coded into four bits. From low order to high order, the categories follow this order: system, owner, group, world. The access code bits within each category are arranged (from low order to high order) as follows: read, write, extend, delete. A bit that is set indicates that the corresponding access is denied.

   The issuing task's UIC is the created region's owner UIC.

   In order to prevent the creation of common blocks that are not easily deleted, the system and owner categories are always forced to have delete access, regardless of the value actually specified in the protection word.

CSRQ$

9.1.12 CSRQ$—Cancel Time Based Initiation Requests

The Cancel Time Based Initiation Requests directive instructs the system to cancel all time-synchronized initiation requests for a specified task, regardless of the source of each request. These requests result from a Run directive.

Fortran Call

CALL CANALL (tsk[,ids])

tsk	task name
ids	directive status

Macro Call

CSRQ$ tsk

tsk	scheduled (target) task name

Macro Expansion

```
CSRQ$   ALPHA
.BYTE   25.,3           ;CSRQ$ MACRO DIC, DPB SIZE=3 WORDS
.RAD50  /ALPHA/         ;TASK ''ALPHA''
```

Local Symbol Definitions

C.SRTN	target task name (4)

DSW Return Codes

IS.SUC	successful completion
IE.INS	task is not installed
IE.PRI	the issuing task is not privileged and is attempting to cancel requests made by another task
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

IE.MAP	the specified system state routine is greater than 4K words from the specified base
IE.ADP	part of the DPB is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid

Notes

1. User mode register contents are preserved across the execution of the kernel mode subroutine. Contents of the user mode registers are passed into the kernel mode registers. Contents of the kernel mode registers are discarded when the subroutine has completed execution.

2. User mode registers appear at the following octal stack offsets when executing the specified subroutine in kernel mode:

   User mode R0 at S.WSR0 Offset on kernal stack
   User mode R1 at S.WSR1 Offset on kernal stack
   User mode R2 at S.WSR2 Offset on kernal stack
   User mode R3 at S.WSR3 Offset on kernal stack
   User mode R4 at S.WSR4 Offset on kernal stack
   User mode R5 at S.WSR5 Offset on kernal stack

   If you wish to return any register values to the user mode registers, you must store the desired values on the stack using the above offsets.

3. Virtual address values passed to system state in a register must be realigned through kernal APR5. For example, if R5 contains address n, and the base virtual address in the DPB is 1000(8), the value in R5 must be aligned using the formula:
   n+120000+base virtual address
   Therefore, the resultant value is n+121000.

4. The system state subroutine should exit by issuing an RTS PC instruction. This causes a successful directive status to be returned as the directive is terminated.

   Caution: Keep in mind that the memory management unit rounds the base address to the nearest 32-word boundary.

TLOG$

9.1.68A TLOG$—Translate Logical Name

The Translate Logical Name directive translates a logical name to its equivalence name, returning the equivalence name string to a specified buffer.

Fortran Call

```
CALL TRALOG (mode,[itbmsk],[dummy],lns,lnssz,iens,ienssz,
                [irsize],[irtbmo],[idsw])
```

mod	The modifier of the logical name within a table
itbmsk	Reserved for future use
dummy	Reserved for future use
lns	Character array containing the logical name string
lnssz	Size (in bytes) of the logical name string
iens	Character array containing the equivalence name string
ienssz	Size (in bytes) of the equivalence name string
irsize	Address of the word to which the size of the resulting equivalence name string is returned
irtbmo	Address of the word to which the table number (low byte) and mode (high byte) of the resulting equivalence string is returned
idsw	Integer to receive the Directive Status Word

Macro Call

```
TLOG$ mod,tbmask,0,lns,lnssz,ens,enssz,rsize,rtbmod
```

mod	The modifier of the logical name within a table
tbmask	Reserved for future use
lns	Character array containing the logical name string
lnssz	Size (in bytes) of the logical name string
iens	Character array containing the equivalence name string
ienssz	Size (in bytes) of the equivalence name string
rsize	Address of the word to which the size of the resulting equivalence name string is returned
rtbmod	Address of the word to which the table number (low byte) and mode (high byte) of the resulting equivalence string is returned

Macro Expansion

```
TLOG$   MOD,TBMASK,0,LNS,LNSSZ,ENS,ENSSZ,RSIZE,RTBMOD
.BYTE   207.,9          ;TLOG$ MACRO DIC, DPB SIZE = 9 WORDS
.BYTE   1               ;SUBFUNCTION CODE FOR TRANSLATION
.BYTE   MOD             ;LOGICAL NAME MODIFIER
.BYTE   TBMASK          ;RESERVED FOR FUTURE USE
.BYTE   0               ;RESERVED FOR FUTURE USE
.WORD   LNS             ;ADDRESS OF LOGICAL NAME BUFFER
.WORD   LNSSZ           ;BYTE COUNT OF LOGICAL NAME STRING
.WORD   ENS             ;ADDRESS OF EQUIVALENCE NAME BUFFER
.WORD   ENSSZ           ;BYTE COUNT OF EQUIVALENCE NAME STRING
.WORD   RSIZE           ;ADDRESS OF BUFFER INTO WHICH EQUIVALENCE
                        ;NAME STRING IS TO BE RETURNED
.WORD   RTBMOD          ;ADDRESS OF BUFFER INTO WHICH TABLE NUMBER
                        ;AND MODIFIER ARE TO BE RETURNED
```

Local Symbol Definitions

T.LENS	Address of equivalence name string (2)
T.LESZ	Byte count of equivalence name string (2)
T.LFUN	Subfunction (1)
T.LLNS	Address of logical name string (2)
T.LLSZ	Byte count of logical name string (2)
T.LMOD	Logical name modifier (1)
T.LRSZ	Buffer address for returned equivalence string
T.LRTM	Buffer address for returned table number and modifier
T.LTBL	Logical table number

DSW Return Codes

IS.SUC	Successful completion.
IE.RBS	The resulting equivalence name string is too large for the buffer to receive it.
IE.LNF	The specified logical name string was not found.
IE.IBS	The length of the logical or equivalence string is invalid. Each string length must be greater than 0 but not greater than 255_{10} characters.
IE.ADP	Part of the DPB or user buffer is out of the issuing task's address space, or the user does not have proper access to that region.
IE.SDP	DIC or DPB size is invalid.

UMAP$

9.1.69 UMAP$—Unmap Address Window

The Unmap Address Window directive unmaps a specified window. After the window has been unmapped, references to the corresponding virtual addresses are invalid and cause a processor trap to occur.

Fortran Call

CALL UNMAP (iwdb[,ids])

iwdb	an 8-word integer array containing a Window Definition Block (see Section 7.5.2.2)
ids	directive status

Macro Call

UMAP$ wdb

wdb	Window Definition Block address

Macro Expansion

```
UMAP$   WDBADR
.BYTE   123.,2          ;UMAP$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   WDBADR          ;WDB ADDRESS
```

Table 9-11
Window Definition Block Parameters

Array Element	*Offset*	*Description*	
Input Parameters			
iwdb(1) bits 0-7	W.NID	ID of the window to be unmapped	
Output Parameters			
iwdb(7)	W.NSTS	Bit settings[17] in the window status word:	
		Bit	*Definition*
		WS.UNM	1 if the window was successfully unmapped

17. If you are a higher-level language programmer, refer to Section 7.5.2 to determine the bit values represented by the symbolic names described.

9.1.71 VRCD$—Variable Receive Data

The Variable Receive Data directive instructs the system to dequeue a variable-length data block for the issuing task; the data block has been queued (FIFO) for the task by a Variable Send Data directive. When a sender task is specified, only data sent by the specified task is received.

Buffer size can be 256_{10} words maximum. If no buffer size is specified, the buffer size is 13_{10} words. If a buffer size greater than 256_{10} is specified, an IE.IBS error is returned.

A 2-word sender task name (in Radix-50 form) and the data block are returned in the specified buffer, with the task name in the first two words. For this reason, the storage you allocate within the buffer should be two words greater than the size of the data portion of the message specified in the directive.

Variable-length data blocks are transferred from the sending task to the receiving task by means of buffers in the secondary pool.

Fortran Call

```
CALL VRCD ([task],bufadr,[buflen][,ids])
```

task	Sender task name
buf	Address of buffer to receive the sender task name and data
buflen	Length of buffer
ids	Integer to receive the Directive Status Word.

If the directive was successful, it returns the number of words transferred into the user buffer. If the directive encounters an error during execution, it returns the error code in the ids parameter.

Any error return of the form IE.XXX is a negative word value. If the status is positive, the value of the status word is the number of words transferred including the task name. For example, if you specify a buffer size of 13 in the VRCD$ call, the value returned in the Directive Status Word is 15 (13 words of data plus the two words needed to return the task name).

Macro Call

```
VRCD$ [task],bufadr[,buflen]
```

task	Sender task name
bufadr	Buffer address
buflen	Buffer size in words

Macro Expansion

```
VRCD$     SNDTSK,DATBUF,BUFSIZ,TI
.BYTE     75.,6            ;VRCD$ MACRO DIC, DPB SIZE=6 WORDS
.RAD50    /SNDTSK/         ;SENDER TASK NAME
.WORD     DATBUF           ;ADDRESS OF DATA BUFFER
```

Note: TI is ignored.

Local Symbol Definitions

R.VDTN	Sender task name (4)
R.VDBA	Buffer address (2)
R.VDBL	Buffer length (2)

DSW Return Codes

IS.SUC	Successful completion.
IE.INS	Specified task not installed.
IE.ITS	No data in task's receive queue.
IE.RBS	Receive buffer is too small.
IE.IBS	Invalid buffer size specified (greater than 255.)
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space
IE.SDP	DIC or DPB size is invalid.

9.1.72 VRCS$—Variable Receive Data Or Stop

The Variable Receive Data Or Stop directive instructs the system to dequeue a variable-length data block for the issuing task; the data block has been queued (FIFO) for the task by a Variable Send Data directive. If there is no such packet to be dequeued, the issuing task is stopped. In this case, another task (the sender task) is expected to issue an Unstop directive after sending the data. When stopped in this manner, the directive status returned is IS.SET, indicating that the task was stopped and that no data has been received; however, since the task must be unstopped in order to see this status, the task can now reissue the Variable Receive Data Or Stop directive to actually receive the data packet.

When a sender task is specified, only data sent by the specified task is received.

Buffer size can be 256_{10} words maximum. If no buffer size is specified, the buffer size is 13_{10} words. If a buffer size greater than 256_{10} is specified, an IE.IBS error is returned.

A 2-word sender task name (in Radix-50 form) and the data block are returned in the specified buffer, with the task name in the first 2 words. For this reason, the storage you allocate within the buffer should be two words greater than the size of the data portion of the message specified in the directive.

Variable-length data blocks are transferred from the sending task to the receiving task by means of buffers in the secondary pool.

FORTRAN Call

```
CALL VRCS ([task],bufadr,[buflen][,ids])
```

task	Sender task name
buf	Address of buffer to receive the sender task name and data
buflen	Length of buffer
ids	Integer to receive the directive status word

If the directive was successful, it returns the number of words transferred into the user buffer. If the directive execution encountered an error, it returns the error code in the ids parameter.

Any error return of the form IE.XXX is a negative word value. If the status is positive, the value of the status word is the number of words transferred including the taskname. For example, if you specify a buffer size of 13 in the VRCS$ call, the value returned in the directive status word is 15 (13 words of data plus the two words needed to return the taskname).

Macro Call

VRCS$ [task],bufadr[,buflen]

task	Sender task name
bufadr	Buffer address
buflen	Buffer size in words

Macro Expansion

```
VRCS$     SNDTSK,DATBUF,BUFSIZ
.BYTE     139.,6           ;VRCS$ MACRO DIC, DPB SIZE=6 WORDS
.RAD50    /SNDTSK/         ;SENDER TASK NAME
.WORD     DATBUF           ;ADDRESS OF DATA BUFFER
.WORD     BUFSIZ           ;BUFFER SIZE
```

Local Symbol Definitions

R.VSTN	Sender task name (4)
R.VSBA	Buffer address (2)
R.VSBL	Buffer length (2)
R.VSTI	Reserved (2)

DSW Return Codes

IS.SUC	Successful completion.
IS.SET	Task was stopped and no data was received.
IE.INS	Specified task not installed.
IE.RBS	Receive buffer is too small.
IE.IBS	Invalid buffer size specified (greater than 255.).
IE.ADP	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	DIC or DPB size is invalid.

9.1.76 WIMP$—What's In My Professional

The What's In My Professional directive is a general purpose system information retrieval mechanism. The directive allows a nonprivileged task to retrieve specific information stored by the system without requiring the task to be mapped to the Executive. In all forms, the WIMP$ directive requires a subfunction, a return buffer, and the return buffer size.

A subfunction specifies the type of information to be returned. The return buffer is space allocated within your task and must be large enough to contain the information that is to be returned. Refer to the descriptions of the implemented subfunction for the specific size of the return buffer.

FORTRAN Call

```
CALL WIMP (SFCN,P1,P2,P3,P4,P5,P6,IDS)
```

MACRO Call

```
WIMP$ SFCN,P1,P2,P3,P4,P5,P6
```

where:

SFCN	subfunction code
P1	parameter 1
P2	parameter 2
P3	parameter 3
P4	parameter 4
P5	parameter 5
P6	parameter 6
IDS	directive status

Macro Expansion

```
WIMP$ SFCN,P1,P2,P3,P4,P5,P6

.BYTE 169.,variable
.WORD SFCN         ;SUBFUNCTION CODE
.WORD P1           ;PARAMETER 1
.WORD P2           ;PARAMETER 2
.WORD Pn           ;PARAMETER n
```

Local Symbol Definitions

G.INSF	SUBFUNCTION CODE (2)
G.IP01	PARAMETER 1 (2)
G.IP02	PARAMETER 2 (2)
G.IP03	PARAMETER 3 (2)
G.IP04	PARAMETER 4 (2)
G.IP05	PARAMETER 5 (2)
G.IP06	PARAMETER 6 (2)

DSW Return Codes

IS.SUC	Successful completion
IE.IDU	Invalid hardware for requested operation
IE.SDP	DIC, DPB size, or subfunction is invalid

Implemented Subfunctions

GI.SSN	Get system serial number

WIMP$ GI.SSN,BUF,SIZ

BUF	Return buffer address
SIZ	Size in words of return buffer (size = 3.)

Output Buffer Format

word 0	high word of system serial number
word 1	middle word
word 2	low word
GI.CFG	Get configuration table

WIMP$ GI.CFG,BUF,SIZ

BUF	Return buffer address
SIZ	Size in words of return buffer (size = 96.)

Table 9-12 lists the offsets in the configuration table as displayed in the user's return buffer. The information contained in the return buffer reflects the current system configuration including hardware and hardware status. Any changes made to the information in the return buffer are not reflected in the system configuration table.

Table 9-12
The Configuration Table Output Buffer Format

Description	*Offset*
Table length in bytes	0
Serial number ROM ID	2
High word of serial number	4
Middle word of serial number	6
Low word of serial number	8.
Number of option slots	10.
Data length of table	12.
Slot 0 ID	14.
Status/error of slot 0	16.
Slot 1 ID	18.
Status/error of slot 1	20.
Slot 2 ID	22.
Status/error of slot 2	24.
Slot 3 ID	26.
Status/error of slot 3	28.
Slot 4 ID	30.
Status/error of slot 4	32.
Slot 5 ID	34.
Status/error of slot 5	36.
Slot 6 ID (not used)	38.
Status/error of slot 6 (not used)	40.
Slot 7 ID (not used)	42.
Status/error of slot 7 (not used)	44.
Keyboard ID (supplied by the keyboard, this could be some other input device)	46.
Keyboard status/error	48.
Base processor type	50.
Base processor status	52.
Primary memory ID	54.
Total system memory size (memory size is value in high order byte in units of 32K words)	56.
Diagnostic ROM version number	58.
Diagnostic ROM error status	60.
Video monitor present	62.
Video monitor status	64.
Audio device ID (not on PRO 325/350)	66.
Audio device status	68.
Keyboard interface ID (2661)	70.
Keyboard interface status/error	72.

Table 9-12 (Cont.)

Description	*Offset*
Printer port interface ID (2661)	74.
Printer port interface status/error	76.
Maintenance port ID	78.
Maintenance port status	80.
Serial comm interface ID	82.
Serial comm interface status/error	84.
Time of day device ID	86.
Time of day status/error	88.
NVR RAM ID	90.
NVR RAM status/error	92.
Floating point ID	94.
Floating point status/error	96.
Interrupt controller ID	98.
Interrupt controller status/error	100.
Reserved locations	102.
Reserved locations	104.
Reserved locations	106.
Reserved locations	108.
Reserved locations	110.
Reserved locations	112.
Reserved locations	114.
Reserved locations	116.
Reserved locations	118.
Reserved locations	120.
Reserved locations	122.
Reserved locations	124.
Reserved locations	126.
Reserved locations	128.
Reserved locations	130.
Reserved locations	132.
Soft restart address	134.
Offset value into boot code	136.
Booted device ID number	138.
Unit number of booted device	140.
Current boot sequence return address	142.
Error flag for ROM diagnostics	144.
Additional information length	146.

CHAPTER 11
DISK DRIVERS

The system's disk drivers support the disks summarized in Table 11-1. Subsequent sections describe these devices and their access in greater detail.

Table 11-1
Standard Disk Devices

Drive	*RPM*	*Sectors*	*Heads*	*Cylinders*	*Bytes/ Drive*	*Blocks/ Drive*
RX50	300	10	2	80/diskette	819,200	800
RD50	3600	16	4	153/surface	5MB	9727
RD51	3600	16	4	306/surface	10MB	19519

11.1 RX50 DESCRIPTION

The RX50 (diskette) subsystem consists of a 5.25-inch dual flexible diskette drive and a separate single-board controller module. The module enables a data processing system to store or retrieve information from any location on one side of each front-loadable diskette.

11.2 RD50 AND RD51 DESCRIPTION

The RD50 and RD51 are Winchester hard disk, multiplatter, random-access devices. They store data in fixed-length blocks on 130mm rigid disk media. Winchester technology uses moving head, noncontact recording. As opposed to the RX50, the RD50 and RD51 storage media cannot be removed from the drive.

11.3 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the information listed in Table 11-2 for disks. A bit setting of 1 indicates that the described characteristic is true for disks.

Table 11-2
Buffer Get LUN Information 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 (devic dependent)
8	1	Device supports 22-bit direct addressing
9	0	Unit software write-locked
10	0	Input spooled device
11	0	Output spooled device
12	0	Pseudo-device
13	0	Device mountable as a communications channel
14	1	Device mountable as a Files-11 volume
15	1	Device mountable

Words 3 and 4 of the buffer contain the maximum logical block number. Note that the high byte of U.CW2 is undefined. The user should clear the high byte in the buffer before using the block number. Word 5 indicates the default buffer size, which is 512 bytes for all disks.

11.4 OVERVIEW OF I/O OPERATIONS

The RX50 and RD50-type disks used on the Professional are Files-11 structured and, therefore, compatible with RMS-11. RMS-11 lets you transfer data between your task and Files-11 structured volumes without having to be concerned with the physical organization of the data on the volume. When transferring data to and from Files-11 volumes, always use the RMS-11 MACRO or high-level language facilities.

The information in this chapter is intended primarily for those cases where you may not want to be restricted to a Files-11 data format and thus cannot use RMS-11. For example,

- ☐ perform I/O with CP/M-strructured RX50 diskettes
- ☐ create a special utility, such as for data backup

The QIO's documented in this chapter let you bypass RMS-11 to handle such cases. Be aware that using QIO's instead of RMS-11 makes program development more complex and increases the chances of error.

Note: Refer to *PRO/RMS-11: An Introduction* for definitions of physical, logical, and virtual blocks, as well as a generic description of disk geometry (sectors, tracks, cylinders).

With some exceptions, the disks supported on the Professional can be accessed in essentially the same manner. Any differences are pointed out in the discussion.

Disks are divided into a series of 256-word blocks, and data is transferred in blocks to and from disks. These transfers may be performed in three possible modes: physical, logical, or virtual. The difference between theses modes is the manner in which the disk is addressed.

11.4.1 Physical I/O Operations

Note: Physical I/O operations are allowed to the RX50 but not to the RD50 or RD51.

In physical I/O operations, data is read or written to the actual sectors (physical blocks) of the disk. No consideration is given to factors such as the interleave of blocks or the track skew, which may be introduced to compensate for any lag of the read/write head. So, physical blocks are numbered consecutively on any one track. Physical blocks are numbered starting with 0.

The RX50 is a single head device. On the RX50, the address of a physical block is expressed as a track and sector address. The first physical block is on track 0, sector 1.

The RD50 and RD51 are multiplatter devices. On these, the address of a physical block is derived by first obtaining the sector number within a track, then the track number within a cylinder, and then the cylinder number. The first physical block is on track 0, sector 0.

11.4.2 Logical I/O Operations

Logical I/O operations transfer data to or from the logically addressable blocks of the disk. Unlike physical blocks, logical blocks are not necessarily contiguous. For the RX50 disk, the sector interleave and track skew are automatically taken into account when a logical READ or WRITE is performed.

Logical blocks are numbered starting with 0.

For the RX50, the first logical block is on track 1, sector 1. The highest-numbered logical block is on track 0, on the highest sector for that track. For the RD50 and RD51, the first logical block is track 0, sector 1.

11.4.3 Virtual I/O Operations

Virtual I/O operations have meaning only within the context of a file. The virtual blocks of a file are numbered consecutively, starting with virtual block 1 which is the start of the file. The consecutively numbered virtual blocks of a file map to logical blocks which may or may not be contiguous. Virtual I/O operations are converted by the file processor in to logical READS and WRITES.

11.5 QIO MACRO

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

Note: If your task is transferring data to a Files-11 structured volume, use RMS-11 to perform I/O operations.

11.5.1 Standard QIO Functions

Table 11-3 lists the standard functions of the QIO macro that are valid for disks.

Table 11-3
Standard QIO Functions for Disks

Format	*Functions*
QIO$C IO.ATT,...	Attach device*
QIO$C IO.DET,...	Detach device
QIO$C IO.KIL,...	Kill I/O**
QIO$C IO.RLB,...,<stadd,size,,blkh,blkl>	READ logical block
QIO$C IO.RPB,...,<stadd,size,,blkh,blkl>	READ physical block***
QIO$C IO.RVB,...,<stadd,size,,blkh,blkl>	READ virtual block
QIO$C IO.WLB,...,<stadd,size,,blkh,blkl>	WRITE logical block
QIO$C IO.WPB,...,<stadd,size,,blkh,blkl>	WRITE physical block***
QIO$C IO.WVB,...,<stadd,size,,blkh,blkl>	WRITE virtual block

* 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. It is highly recommended that you do not attach any volumes as this will prevent the mount/dismount mechanisms from working and possibly lead to either a program "hang" or a system "hang".

* *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.

* **Not supported for RD50 or RD51. Supported only for RX50.

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).
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.

To perform physical or logical I/O operations with QIO's, your task must be privileged.

IO.RVB and IO.WVB are associated with file operations. 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.

Use of the QIO virtual I/O operations requires caution as it is possible to quickly exhaust system ressources. Simply writing and reading files with QIO's is not sufficient. The file must be extended with $EXTEND calls to RMS-11 or the end of file pointer will not be moved. Reading and writing a file via QIO's without properly setting the FB$SHR field in the RMS-11 FAB block may cause system hangs when system free space is exhausted.

> Note: If you are using FCS instead of RMS-11, 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.

Each disk driver supports the subfunction bit IQ.X: inhibit retry attempts for error recovery. The subfunction bit is used by ORing it into 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.

11.6 STATUS RETURNS

The error and status conditions listed in Table 11-4 are returned by the disk drivers described in this chapter.

Table 11-4
Disk Status Returns

Code	*Reason*
IS.SUC	*Successful completion* The operation specified in the QIO directive was completed successfully. The second word of the I/O status block can be examined to determine the number of bytes processed, if the operation involved reading or writing.
IS.PND	*I/O request pending* The operation specified in the QIO directive has not yet been executed. The I/O status block is filled with 0s.
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 illegal logical block number was specified.
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*ql; 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 directive was not ready to perform the desired I/O operation.
IE.FHE	*Fatal hardware error* The controller is physically unable to reach the location where input/output operation is to be performed. The operation cannot be completed.
IE.IFC	*Illegal function* A function code was specified in an I/O request that is illegal for disks.
IE.MII	*Media inserted incorrectly* The controller has detected that the media (such as a floppy diskette) was not inserted correctly. To correct the problem, reinsert the media properly.
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.

Table 11-4 (Cont.)
Disk 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.
IE.OVR	Illegal read overlay request A read overlay was requested, and the physical device unit specified in the QIO directive was not the physical device unit from which the task was installed. The read overlay function can only be executed on the physical device unit from which the task image containing the overlays was installed.
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. Alternately, 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.
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, the system 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.

CHAPTER 12
THE TERMINAL DRIVER

12.1 INTRODUCTION

The system supports a single full-duplex terminal driver which includes the following features:

- ☐ Full-duplex operation
- ☐ Type-ahead buffering
- ☐ Eight-bit characters
- ☐ Transparent read and write
- ☐ Formatted read and write
- ☐ Read after prompt
- ☐ Read with no echo
- ☐ Read with special terminator
- ☐ Optional time-out on solicited input
- ☐ Device-independent cursor control

12.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the information noted in Table 12-1 for terminals. A setting of 1 indicates that the described characteristic is true for terminals.

Table 12-1
Buffer Get LUN Information 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
8	0	Device supports 22-bit direct addressing
9	0	Unit software write-locked
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).

12.3 QIO MACRO

Table 12-2 lists the standard and device-specific functions of the QIO macro that are valid for terminals.

Table 12-2
Standard and Device-Specific QIO Functions for Terminals

Format	*Function*
Standard Functions:	
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[,tmo]>	READ logical block (read typed input into buffer).
QIO$C IO.RVB,...,<stadd,size[,tmo]>	READ virtual block (read typed input into buffer).
QIO$C IO.WLB,...,<stadd,size,vfc>	WRITE logical block (print buffer contents).
QIO$C IO.WVB,...,<stadd,size,vfc>	WRITE virtual block (print buffer contents).
Device-Specific Functions:	
QIO$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 SF.GMC,...,<stadd,size>	GET multiple characteristics
QIO$C IO.GTS,...,<stadd,size>	GET terminal support
QIO$C IO.RAL,...,<stadd,size[,tmo]>	READ logical block, pass all bits
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 terminator
QIO$C SF.SMC,...,<stadd,size>	SET multiple characteristics.
QIO$C IO.WAL,...,<stadd,size,vfc>	WRITE logical block, pass all bits
QI0$C IO.WBT,...,<stadd,size,vfc>	WRITE logical block, break through most I/O conditions at terminal
QIO$C IO.WSD,...,<stadd,size,,type>	WRITE special data
QIO$C IO.RSD,...,<stadd,size,tmo,type>	READ special data

ast	The entry point for an unsolicited- input-character AST.
parameter 2	A number that can be used to identify this terminal as the input source upon entry to an unsolicited character AST routine.
ast2	The entry point for an INTERRUPT/DO sequence AST. (See Section 12.5.2)
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. The buffer must be within the task's address space.
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. The buffer must be within the task's address space. For SF.GMC, IO.GTS, and SF.SMC functions, size must be an even value.
stadd	The starting address of the data buffer. The address must be word aligned for SF.GMC, IO.GTS, and SF.SMC, IO.RSD, IO.WSD ; otherwise, stadd may be on a byte boundary.
table	The address of the 16-word special terminator table.
tmo	An optional time-out count in 10-second intervals. If 0 is specified, no time-out can occur. Time-out is the maximum time allowed between two input characters before the read is aborted.
type	The data type of the buffer contents.
vfc	A character for vertical format control from Table 12-12 (Vertical Format Control Characters)

12.3.1 Subfunction Bits

Most device-specified functions supported by terminal drivers described in this section are selected using ''subfunction bits.'' One or more functions can be selected by ORing their relative bits in a QIO function. Table 12-4 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 in Table 12-3.

Table 12-3
Definition of Subfunction Bits

Symbolic Name	*Subfunction*
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; unsolicited characters are stored in the type-ahead buffer until they are read by the task
TF.RAL	Read all bits
TF.RCU	Restore cursor position
TF.RNE	Read with no echo
TF.RST	Read with special terminators
TF.TMO	Read with time-out
TF.WBT	Break through write
TF.WAL	Write all bits
TF.XCC	Send an INTERRUPT/DO sequence to the P/OS Dispatcher.

Table 12-4 lists subfunction bits that can be ORed with QIO functions. Additional details for using subfunction bits are included in Section 12.3.2.

If a task invokes a subfunction bit that is not supported on the system, the subfunction bit is ignored, but the QIO request is not rejected.

The following example is a QIO request using more than one subfunction bit: a nonechoed (TF.RNE) read, 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>
```

12.3.2 Device-Specific QIO Functions

All functions except SF.GMC, IO.RPR, SF.SMC, IO.RTT, and IO.GTS can be issued by ORing a particular subfunction bit with another QIO function. These subfunction bits are specified in the following descriptions; subfunction bits are described in general in Section 12.3.1.

In addition to the device-specific QIO functions, this section also describes the use of subfunction bits TF.ESQ and TF.BIN.

Table 12-4
Summary of Subfunction Bits

		Allowed Subfunction Bits											
Function	*Equivalent Subfunctions*	*TF.AST*	*TF.BIN*	*TF.CCO*	*TF.ESQ*	*TF.NOT*	*TF.RAL*	*TF.RCU*	*TF.RNE*	*TF.RST*	*TF.TMO*	*TF.WAL*	*TF.XCC*
Standard Functions													
IO.ATT		X			X								
IO.DET													
IO.KIL													
IO.RLB							1	X	1	X			
IO.RVB							2			2	2		
IO.WLB				X				X					
IO.WVB				2				2				2	
Device-Specific Functions													
IO.ATA	IO.ATT!TF.AST				X	X							X
IO.CCO	IO.WLB!TF.CCO											3	
SF.GMC													
IO.GTS													
IO.RAL	IO.RLB!TF.RAL								X	1	X		
IO.RNE	IO.RLB!TF.RNE						1			1	X		
IO.RPR			X				1		X	1	X		
IO.RST	IO.RLB!TF.RST						1		X		X		
IO.RTT						1		X		X			X
SF.SMC													
IO.WAL	IO.WLB!TF.WAL			3				3					
IO.WBT	IO.WLB!TF.WBT			X				X				3	
IO.WSD													
IO.RSD											X		

Notes:
1. Exercise great care when using Read All and Read with Special Terminators together. Obscure problems can result.
2. These subfunctions are allowed but are not effective. They are stripped off when the read or write virtual operation is converted to a read or write logical operation.
3. 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.

12.3.2.1 IO.ATA—IO.ATA is a variation of the Attach function. The use of this function is eased by the addition of TF.NOT and TF.XCC subfunction bits, described later in this section. IO.ATA specifies asynchronous system traps (ASTs) to process unsolicited input characters. When called as follows:

```
QIO$C IO.ATA,...,<[AST],[PARAMETER2][,AST2]>
```

Note: A minimum of one AST parameter (ast or ast2) is required.

This function attaches the terminal and identifies "ast" and "ast2" as entry points for an unsolicited-input-character AST. Control passes to ast whenever an unsolicited character (other than CTRL/Q, CTRL/S, CTRL/X, or CTRL/O) is input. If the ast2 parameter is specified, an INTERRUPT/DO sequence results in the specified AST being entered in that parameter. If ast2 is not specified, an INTERRUPT/DO sequence results in the specified AST being entered in the ast parameter.

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

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

Upon entry to the AST routines, the unsolicited character and parameter 2 are in the top word on the stack, as shown below. That word must be removed from the stack before exiting the AST.

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; parameter 2, in the high byte, is a user-specified value that can be used to identify individual terminals in a multiterminal environment

The processing of unsolicited input ASTs is eased through the use of TF.NOT and TF.XCC subfunction bits. When TF.XCC is included in the IO.ATA function, all characters (except INTERRUPT/DO sequences) are handled in the manner previously described. INTERRUPT/DO sequences cause the P/OS Dispatcher to abort the application.

When unsolicited terminal input (except an INTERRUPT/DO sequence) is received by the terminal driver and the TF.NOT subfunction is used, 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 (parameter 2). 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.

See Chapter 5 for further details on ASTs.

IO.ATA is equivalent to IO.ATT ORed with the subfunction bit TF.AST.

12.3.2.2 IO.ATT!TF.ESQ—The task issuing this function attaches a terminal and notifies the driver that it recognizes escape sequences input from that terminal. Escape sequences are recognized only for solicited input. (See Section 12.6 for a discussion of escape sequences.)

If the terminal has not been declared capable of generating escape sequences, IO.ATT!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 SF.SMC function is used to declare the terminal capable of generating escape sequences (see Table 12-5 Driver-Terminal Characteristics for SF.GMC and SF.SMC Functions).

12.3.2.3 IO.CCO—This write function directs the driver to write to the terminal regardless of a CTRL/O condition that may be in effect. If CTRL/O is in effect, it is cancelled before the write is done.

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

12.3.2.4 SF.GMC—The Get Multiple Characteristics function returns terminal characteristics information, as follows:

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

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 12-5. 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.

Table 12-5
Driver-Terminal Characteristics for SF.GMC and SF.SMC Functions

Bit Name	*Octal Value*	*Meaning (if asserted)*
TC.ACR	24	Wrap-around mode
TC.BIN	65	Binary input mode (read-pass-all) no characters are interpreted as control characters.
TC.CTS	72	Suspend output to terminal 0 = resume 1 = suspend
TC.EPA *	40	Parity sense, valid only when TC.PAR is enabled 0 = odd parity 1 = even parity
TC.ESQ	35	Input escape sequence recognition
TC.FDX	64	Full-duplex mode
TC.HFF	17	Hardware form-feed capability (If 0, form-feeds are simulated using TC.LPP.)
TC.HFL	13	Number of fill characters to insert after a RETURN (0-7=x)
TC.HHT	21	Horizontal tab capability (if 0, horizontal tabs are simulated using spaces.)
TC.LPP	2	Page length (1-255.=x)
TC.NEC	47	Echo suppressed
TC.PAR *	37	Enable and disable parity 1 = enabled 0 = disabled
TC.RSP *	3	Receiver speed (bits-per-second) (see Table 12-7)
TC.SCP	12	Terminal is a scope (CRT)
TC.SMR	25	Upper-case conversion disabled
TC.TBF	71	Type-ahead buffer count (read), or flush (write)
TC.TTP	10	Terminal type (=0-255.=x) (see Table 12-6)
TC.VFL	14	Send four fill characters after line feed
TC.WID **	1	Page width (=1-255.=x)
TC.XSP *	4	Transmitter speed (bits-per-second) (see Table 12-7)
TC.8BC	67	Pass eight bits on input, even if not binary input mode (TC.BIN)

* Applies to TT2: only. TT1: cannot set this attribute.

* *Unsolicited input that fills the buffer before a terminator is received is likely invalid. When this happens, the driver discards the input by simulating a CTRL/U and echoing ^U.

For the TC.TTP characteristic (terminal type), one of the values shown in Table 12-6 is returned in the high byte.

The TC.TTP characteristic, when read by the terminal driver, sets implicit values for terminal characteristics TC.LPP, TC.WID, TC.HFF, TC.HHT, TC.VFL, and TC.SCP as shown in Table 12-6. These values can be changed (overridden) by subsequent Set Multiple Characteristics requests. In addition, TC.TTP is used by the terminal driver to determine cursor positioning commands, as appropriate.

Table 12-6
TC.TTP (Terminal Type) Values Set by SF.SMC and Returned by SF.GMC

		Implicit Characteristics[*]						
Symbolic	*Terminal Type*	*TC.LPP*	*TC.WID*	*TC.HFF*	*TC.HHT*	*TC.HFL*	*TC.VFL*	*TC.SCP*
T.UNK0	Unknown							
T.V100	VT100	24	80		1			1
T.L120	LA120	66	132	1				
T.LA12	LA12	66	132	1	1			
T.L100	LA100	66	132	1	1			
T.V101	VT101	24	80		1			1
T.V102	VT102	24	80		1			1
T.V105	VT105	24	80		1			1
T.V125	VT125	24	80		1			1
T.LQP2	LQP02	66	80	1	1			
T.LA50	LA50	66	80	1	1			
T.BMP1	PC300[**]	24	80					1

[*] Implicit characteristics are shown as supported by the driver. Values not shown are not automatically set by the driver. An "unknown" terminal type has no implicit characteristics.

[**]The PC300 Series Bitmap Display is the default terminal for the Professional.

The TC.CTS characteristic returns the present suspend (CTRL/S), resume (CTRL/Q), or suppress (CTRL/O) state set via the SF.SMC function. Values returned are as follows:

Value Returned	*State*
0	Resume (CTRL/Q)
1	Suspend (CTRL/S)
2	Suppress (CTRL/O)
3	Both suppress and suspend

When a value of 0 is used with the SF.SMC function, the suspend state is cleared; a value of 1 selects the suspend state.

Note: If you stop output to the terminal screen by pressing the HOLD key on a PC300, TC.CTS does not indicate that output has stopped. In addition, if you stop output to the terminal screen by pressing the NO SCROLL key on a VT100 series terminal or the HOLD key on a PC300 series terminal, output cannot be resumed with TC.CTS.

The TC.TBF characteristic 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, the value returned can be used to read the exact number of characters typed, rather than a typical value of 80_{10} or 132_{10} characters for the terminal.

Note:

1. The maximum capacity of the type-ahead buffer is 36. characters.

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

Table 12-7 lists the values for terminal receiver and transmitter speeds (TC.RSP and TC.XSP).

Table 12-7
Receiver and Transmitter Speed Values (TC.RSP, TC.XSP)

TC.RSP or TC.XSP Value value	*Actual Baud Rate (bits per second)*
S.0 = 1.	
S.50 = 2.	50
S.75 = 3.	75
S.100 = 4.	100
S.110 = 5.	110
S.134 = 6.	134
S.150 = 7.	150
S.200 = 8.	200
S.300 = 9.	300
S.600 = 10.	600
S.1200 = 11.	1200
S.1800 = 12.	1800
S.2000 = 13.	2000
S.2400 = 14.	2400
S.3600 = 15.	3600
S.4800 = 16.	4800
S.7200 = 17.	7200
S.9600 = 18.	9600
S.19.2 = 21.	19200

12.3.2.5 IO.GTS—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.

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 12-8)

T.xxxx (Table 12-6)

TC.xxx (Table 12-5)

The SE.xxx error returns described in Table 12-9.

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.

Table 12-8
Information Returned by Get Terminal Support (IO.GTS) QIO

Mnemonic	*Meaning When Set to 1*
Word 0 of Buffer:	
F1.ACR	Automatic CR/LF on long lines
F1.BUF	Checkpointing during terminal input
F1.UIA	Unsolicited-input-character AST
F1.CCO	Cancel CTRL/O before writing
F1.ESQ	Recognize escape sequences in solicited input
F1.LWC	Lower- to uppercase conversion
F1.RNE	Read with no echo
F1.RPR	Read after prompting
F1.RST	Read with special terminators
F1.RUB	CRT rubout
F1.TRW	Read all and write all
F1.UTB	Input characters buffered in task's address space
F1.VBF	Variable-length terminal buffers
Word 1 of Buffer:	
F2.SCH	Set characteristics QIO (SF.SMC)
F2.GCH	Get characteristics QIO (SF.GMC)
F2.SFF	Formfeed can be simulated
F2.CUP	Cursor positioning
F2.FDX	Full Duplex Terminal Driver

12.3.2.6 IO.RAL—The Read All function causes the driver to pass all bits to the requesting task. The driver does not intercept control characters or mask out the high-order bit. For example, CTRL/Q, CTRL/S, CTRL/O, and CTRL/Z and INTERRRUPT/DO sequences are passed to the program and are not interpreted by the driver.

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

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

12.3.2.7 IO.RNE—The IO.RNE function reads terminal input characters without echoing the characters back to the terminal for immediate display. This feature can be used 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 12-5, bit TC.NEC.)

The IO.RNE function is equivalent to IO.RLB ORed with the subfunction bit TF.RNE.

12.3.2.8 IO.RPR—The IO.RPR Read After Prompt functions as an IO.WLB (to write a prompt to the terminal) followed by IO.RLB. However, IO.RPR differs from this combination of 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 IO.WAL/IO.RLB is used, 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 ORed with IO.RPR to write the prompt as a Write All (TF.BIN). In addition, read subfunction bits TF.RAL, TF.RNE, and TF.RST can be used with IO.RPR.

12.3.2.9 IO.RPR!TF.BIN—This function results in a read after a "binary" prompt; that is, a prompt is written by the driver with no character interpretation (as if it were issued as an IO.WAL).

12.3.2.10 IO.RST—This function is similar to an IO.RLB, except 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, with certain exceptions.* For example, a horizontal TAB (011) is not expanded, a RUBOUT (or DEL, 177) does not erase.

* If upper- 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.

IO.RST is equivalent to IO.RLB!TF.RST.

12.3.2.11 SF.SMC—This function enables a task to set and reset the characteristics of a terminal. Set Multiple Characteristics is the inverse function of SF.GMC. Like SF.GMC, it is called in the following way:

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

stadd	The starting address of a buffer of length "size" bytes. Each word in the buffer has the form .BYTE characteristic-name .BYTE value
characteristic-name	One of the symbolic bit names given in Table 12-5.
value	Either 0 (to clear a given characteristic) or 1 (to set a characteristic).

Table 12-8 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 12-6.

Specifying any value for TC.TBF flushes (clears) the type-ahead buffer (forces the type-ahead buffer count to 0).

12.3.2.12 IO.RTT—This QIO function reads characters in a manner like the IO.RLB function, except a user-specified character terminates the read operation. The specified character's code can range from 0–377. It is user designated by setting the appropriate bit in a 16-word table that corresponds to the desired character. Multiple characters can be specified by setting their corresponding bits.

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, since the third word of the table contains bits representing character codes 40–57.

If the CTRL/S (023), CTRL/Q (021), and/or any characters whose codes are greater than 177 are desired as the terminator character(s), the terminal must be set to read-pass-all operation (TC.BIN=1), or read-pass 8-bits (TC.8BC), as listed in Table 12-5.

The optional time-out count parameter can be included, as desired.

12.3.2.13 IO.WAL—The Write All function 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 wrap-around has been selected.

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

12.3.2.14 IO.WBT—The IO.WBT function 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 break-through write was effected (if the terminal is not in the full-duplex mode).
- ☐ CTL/O, if in effect, 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. Break-through write may only be issued by a privileged task.

12.3.2.15 IO.WSD—The Write Special Data function is used to communicate nontext information to the terminal task. The buffer address and length are the same as for IO.WLB. The data type parameter indicates to the terminal task what type of data is contained in the buffer. The data type is:

SD.GDS	PRO/GIDIS output

Note: This QIO implements a data path to the terminal subsystem that is also used by the CORE Graphics Library and by the VT125 (ReGIS) terminal emulator within the PRO/Communications application.

12.3.2.16 IO.RSD—The Read Special Data function is also used in communicating nontext information to the terminal subsystem. The buffer address, length, and timeout are the same as for IO.RLB. The data type parameter indicates to the terminal what type of data is to be read.

Note: This QIO implements a data path to the terminal subsystem that is also used by the CORE Graphics Library and by the VT125 (ReGIS) terminal emulator within the PRO/Communications application.

The following restrictions apply to the use of IO.RSD:

1. In some ways, IO.RSD is the same as a normal read. One result of this is that if there is a read currently oustanding to the keyboard, the IO.RSD does not take effect until the read to the keyboard is complete.

2. While an IO.RSD is pending, no input processing takes place until it completes. So any characters that come in from the keyboard go directly to the typeahead buffer, no ASTs take place, and no characters are echoed.
3. When special data comes into the terminal driver from the terminal task (for example, a PRO/GIDIS report) and no IO.RSD is outstanding, the special data goes into a special typeahead buffer. That typeahead buffer is capable of holding a maximum of 36 bytes. If more characters are input than the buffer can hold, those characters are discarded and no error message is returned.

   If there is an IO.RSD pending when special data comes into the terminal driver from the terminal task, the data goes directly into a read buffer. However, the length of one report may not exceed 36 bytes.

As a result of these restrictions, the recommended way to get a special data report is to first issue the IO.WSD to cause the report to occur and then to issue an IO.RSD for the exact length of the request. This causes IO.RSD to complete immediately, preventing it from blocking the keyboard input.

12.4 STATUS RETURNS

Table 12-9 lists error and status conditions that are returned by the terminal driver to the I/O status block.

Most error and status codes returned are byte values. For example, the value for IS.SUC is 1. However, IS.CC, IS.CR, IS.ESC, and IS.ESQ are word values. When any of these codes are returned, the low byte indicates successful completion, and the high byte shows what type of completion occurred.

To test for one of these word-value return codes, first test the low byte of the first word of the I/O status block for the value IS.SUC. Then, test the full word for IS.CC, IS.CR, IS.ESC, IS.ESQ, or IS.CSQ. (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 since 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 as described in Sections 12.3.2.4 and 12.3.2.11. 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 12-9
Terminal Status Returns

Code	*Reason*
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.
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.CC	*Successful completion on a read* The line of input read from the terminal was terminated by an INTERRUPT/DO sequence. 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 RETURN. The input buffer contains the bytes read.
IS.ESC	*Successful completion on a read* The line of input read from the terminal was terminated by an Escape 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 zeroes.
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.
IE.ABO	*Operation aborted* The specified I/O operation was cancelled 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.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.
IE.DNA	*Device not attached* The physical device unit specified in an IO.DET function was not attached by the issuing task. This code has no bearing on the attachment status of other tasks.

Table 12-9 (Cont.)

Code	*Reason*
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 that a time-out occurred on the physical device unit (that is, an interrupt was lost).
IE.IES	*Invalid escape sequence* An escape sequence was started but escape-sequence syntax was violated before the sequence was completed. (See Section 12.6.4.) 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 illegal for terminals.
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.
IE.PES	*Partial escape sequence* An escape sequence was started, but read-buffer space was exhausted before the sequence was completed. See Section 12.6.4.3.
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, a byte count of 0 was specified, or an odd or 0 AST address was specified.
SE.NIH	A terminal characteristic other than those in Table 12-5 was named in an SF.GMC or SF.SMC request, or a task attempted to assert TC.PRI.
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.VAL	The new value specified in an SF.SMC request for the TC.TTP terminal characteristic was not one of those listed in Table 12-6.

12.5 CONTROL CHARACTERS AND SPECIAL KEYS

This section describes the meanings of the system's special terminal control characters and keys. Note that the driver does not recognize control characters and special keys during a Read All request (IO.RAL), and recognizes only some of them during a Read with Special Terminators (IO.RST).

12.5.1 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 12-10, CTRL/R, CTRL/U, and CTRL/Z, are echoed on the terminal as ^R, ^U, and ^Z, respectively.

Note: The use of control characters on PC 300 Series machines is not recommended except when the PC 300s are connected to another system as a terminal. In normal circumstances use the function keys on the PC300s.

Table 12-10
Terminal Control Characters

Character	*Meaning*
CTRL/O	CTRL/O suppresses terminal output. 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. For unattached terminals, CTRL/O suppresses output for only the current output buffer (typically one line).
CTRL/Q	CTRL/Q resumes terminal output previously suspended by means of CTRL/S.
CTRL/S	CTRL/S suspends terminal output. (Output can be resumed by typing CTRL/Q.)
CTRL/R	Typing CTRL/R results in a 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 tabs and/or rubouts in an input line. CTRL/R is also useful for CRT terminals for the CRT rubout. 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	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 RETURN and a line feed.
CTRL/X	This character clears the type-ahead buffer.
CTRL/Z	CTRL/Z indicates an end-of-file for the current terminal input. Note: On the PC300 series systems, the Hold Screen key should be used instead of CTRL/S and CRTL/Q.

Note: On the PC300 series systems, the HOLD key should be used instead of CTRL/S and CRTL/Q.

12.5.2 INTERRUPT/DO AST Information

If the application has done an IO.ATA QIO without specifying the TF.XCC subfunction, the following will happen:

Key One	*Key Two*	*Result*
non Interrupt key	Do key	The non-Interrupt key is handled as usual and the application gets the escape sequence for the Do key
Interrupt key	non Do key	The Interrupt key is discarded and the non Do key is handled as if Interrupt had not been pressed

Interrupt key	Do key	Applications AST routine is activated just as if a CTRL/C was typed
CTRL/C		Applications AST routine is activated as for RSX-11M-PLUS

If the application has not done an IO.ATA QIO or if it has done one with the TF.XCC subfunction bit set, the following will happen:

Key One	*Key Two*	*Result*
non Interrupt key	Do key	The non-Interrupt key is handled as usual and the application gets the escape sequence for the Do key
Interrupt key	non Do key	The Interrupt key is discarded and the non Do key is handled as if Interrupt had not been pressed
Interrupt key	Do key	The PRO/Dispatcher is notified; aborts all application tasks. The application gets no indication that anything happened
CTRL/C		The PRO/Dispatcher is notified; the application gets no indication that anything happened

If an application puts the terminal in Read Pass All mode or if it specifies TF.RAL on a read, all keys, except for Hold Screen and Print Screen, will go into the type-ahead buffer unprocessed. The Interrupt and Do keys will go in as the escape sequences that they represent.

Any characters for which there is no read or AST request outstanding will be put into the type-ahead buffer. The buffer is 36 bytes long. If the buffer is full, and if a characer is typed that would go into the type-ahead buffer, a bell is echoed and the character is discarded. When either CTRL/C or the INTERRUPT/DO sequence is entered, the type-ahead buffer is flushed.

12.5.3 Special Keys

The RETURN, and DELETE keys have special significance for terminal input, as described in Table 12-11. A line can be terminated by a 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.

12.6 ESCAPE SEQUENCES

Escape sequences are strings of two or more characters beginning with an ESC (033) character.

Escape sequences provide a way to pass input to a task without interpretation by the operating system. This could be done with a number 1-character Read All functions, but escape sequences allow them to be read with IO.RLB requests.

Table 12-11
Special Terminal Keys

Key	*Meaning*
RETURN	Typing RETURN terminates the current line and causes the carriage or cursor to return to the first column on the line.
DELETE	Typing DELETE 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.
	DELETE 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 are 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-return option, and a DELETE erases the last character of a previous line, the cursor is not moved to the previous line.

12.6.1 Definition

The format of an escape sequence as defined in American National Standard X 3.41 - 1974 and used in the VT100 is:

ESC ... F

ESC	The introducer control character (33(8)) that is named escape.
...	The intermediate bit combinations that may or may not be present. I characters are bit combination 40(8) to 57(8) inclusive in both 7- and 8-bit environments.
F	The final character. F characters are bit combinations 60(8) to 176(8) inclusive in escape sequences in both 7- and 8-bit environments.

The occurrence of characters in the inclusive ranges 0(8) to 37(8) is technically an error condition whose recovery is to execute immediately the function specified by the character and then continue with 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(8)) or the character SUB (32(8)) occurs, the current escape sequence is aborted, as is the case with any control character.

12.6.2 Prerequisites

Two prerequisites must be satisfied before escape sequences can be received by a task.

1. The task must "ask" for them by issuing an IO.ATT function and invoking the subfunction bit TF.ESQ.
2. The terminal must be declared capable of generating escape sequences. A way to tell the driver that the terminal can generate escape sequences is by issuing the Set Multiple Characteristics request. (See Section 12.3.2.11).

If these prerequisites are not satisfied, the ESC character is treated as a line terminator.

12.6.3 Characteristics

Escape sequences always act as line terminators. That is, an input buffer may contain other characters that are not part of an escape sequence, but an escape sequence always comprises the last characters in the buffer.

Escape sequences are not echoed. However, if a non-CRT DELETE sequence is in progress, it is closed with a backslash when an escape sequence is begun.

Escape sequences are not recognized in unsolicited input streams. Neither are they recognized in a Read with Special Terminators (subfunction bit TF.RST) nor in a Read All (subfunction bit TF.RAL).

12.6.4 Escape Sequence Syntax Violations

A violation of the syntax defined in Section 12.6.1 causes the driver to abandon the escape sequence and to return an error (IE.IES).

12.6.4.1 DEL (177)—The character DELETE is not legal within an escape sequence. If it occurs at any point within an escape sequence, the entire sequence is abandoned and deleted from the input buffer.

12.6.4.2 Control Characters (0–037)—The reception of any character in the range 0 to 037 (with four exceptions is a syntax violation that terminates the read with an error (IE.IES). Four control characters are allowed: 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.

12.6.4.3 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 the following characters are entered:

```
ESC ! A
```

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

```
        IOSB        IE.PES
                      2
```

The "A" is treated as unsolicited input.

12.7 VERTICAL FORMAT CONTROL

Table 12-12 is a summary of all characters used 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 12-12
Vertical Format Control Characters

Octal Value	*Character*	*Meaning*
040	blank	SINGLE SPACE—Output one line feed, print the contents of the buffer, and output a RETURN. Normally, printing immediately follows the previously printed line.
060	0	DOUBLE SPACE—Output two line feeds, print the contents of the buffer, and output a RETURN. Normally, the buffer contents are printed two lines below the previously printed line.
061	1	PAGE EJECT—If the terminal supports FORM FEEDs, output a form feed, print the contents of the buffer, and output a 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—Print the contents of the buffer and output a RETURN, normally overprinting the previous line.
044	$	PROMPTING OUTPUT—Output one line feed and print the contents of the buffer. This mode of output is intended 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—Print the buffer contents without addition of vertical format control characters.

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

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

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 will not back up to the previous line.

12.8 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 type-ahead buffer is used 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:

   - ☐ 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. A character is not echoed when it is stored in the buffer. Echoing a character is deferred until it is retrieved from the buffer, since the read mode (for example, read-without-echo) is not known by the driver until then.

   Note: Depending on the terminal mode and the presence of a read function, read subfunctions and an unsolicited input AST, the INTERRUPT/DO, CTRL/O, CTRL/Q, CTRL/S, and CTRL/X characters may be processed immediately and not stored in the type-ahead buffer.

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 or slaved.

3. **Flush the buffer:**

   The buffer is flushed (cleared) when:

   - ☐ CTRL/X is received.
   - ☐ INTERRUPT/DO is received.

   Note:
   Exceptions: CTRL/X does 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.

12.9 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 only performed with the line in an idle state (not executing a read or a write request).

12.10 INTERMEDIATE INPUT AND OUTPUT BUFFERING

Input buffering for checkpointable tasks with checkpointing enabled is provided in the terminal driver 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 as long as the type-ahead buffer is not full.

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 two purposes:

1. It reduces time spent at system state.
2. It enables task checkpointing during the transfer to the terminal (if all output fits in one buffer list).

12.11 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 is used to define 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 will 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, the TF.RCU subfunction can be used 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.

12.12 PROGRAMMING HINTS

Using IO.WVB instead of IO.WLB is recommended when writing to a terminal. If the write actually goes to a terminal, the Executive converts the IO.WVB to an IO.WLB request. However, if the LUN has been redirected to an appropriate device—a disk, for example—the use of an IO.WVB function will be rejected because a file is not open on the LUN. This prevents privileged tasks from overwriting block zero of the disk.

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

CHAPTER 13
THE XK COMMUNICATIONS DRIVER

13.1 INTRODUCTION

The Professional 300 XK communications driver (XK driver) permits use of the XK communication port in asynchronous mode. The XK driver provides the following features:

- ☐ Full duplex operation
- ☐ Input buffering
- ☐ Unsolicited event AST's
- ☐ Transfer length of up to 8128 bytes
- ☐ Optional time-out on solicited input
- ☐ Optional XON/XOFF support
- ☐ Modem support

13.2 GET LUN INFORMATION MACRO

Word 2 of the buffer filled by the Get LUN Information system directive (the first characteristics word) contains the information noted in Table 13-1 for the XK driver. A setting of 1 indicates that the described characteristic is true.

Table 13-1
Buffer Get LUN Information for XK Driver

Bit	*Setting*	*Meaning*
0	0	Record-oriented device
1	0	Carriage-control device
2	0	Terminal device
3	0	File structured device
4	0	Single-directory device
5	1	Sequential device
6	0	Mass storage device
7	0	User-mode diagnostics supported, device dependent
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 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 size of the internal input ring buffer.

13.3 QIO MACRO

Table 13-2 lists the standard and device-specific functions of the QIO macro that are valid for the XK driver.

Table 13-2
Standard and Device-Specific QIO Functions

Format		*Function*
Standard Functions		
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[,tmo]>	Read logical block (read input into buffer).
QIO$C	IO.RVB,...,<stadd,size[,tmo]>	Read virtual block (read input into buffer).
QIO$C	IO.WLB,...,<stadd,size,vfc>	Write logical block (send contents of buffer).
QIO$C	IO.WVB,...,<stadd,size,vfc>	Write virtual block (send contents of buffer).

Table 13-2 (Cont.)

Format		*Function*
Device-Specific Functions		
QIO$C	IO.ANS,...,<stadd,size>	Initiate a connection in answer mode, either in response to a ringing line, or if a connection already exists.
QIO$C	IO.ATA,...,<ast[,par2]>	Attach device, specify unsolicited event AST.
QIO$C	IO.BRK,...,<type>	Send a BREAK.
QIO$C	IO.CON,...,<stadd,size[,tmo]>	Dial and connect.
QIO$C	SF.GMC,...,<stadd,size>	Get multiple characteristics.
QIO$C	IO.HNG,...	Hang up a line.
QIO$C	IO.LTI,...,<stadd,size[,par3]>	Connect for unsolicited event AST's while detached.
QIO$C	IO.ORG,...,<stadd,size[,tmo]>	Initiate a connection in originate mode, assuming the line has already been connected.
QIO$C	IO.RAL,...,<stadd,size[,tmo]>	Read logical block, pass all bits.
QIO$C	IO.RNE,...,<stadd,size[,tmo]>	Read logical block, do not echo.
QIO$C	SF.SMC,...,<stadd,size>	Set multiple characteristics.
QIO$C	IO.TRM,...,<stadd,1>	Unload driver.
QIO$C	IO.UTI,...	Disable unsolicited event AST's while detached.
QIO$C	IO.WAL,...,<stadd,size>	Write logical block, pass all bits.

ast	The entry point for an unsolicited event AST.
par2 par3	A number that can be used to identify this line as the input source upon entry to an unsolicited event AST routine
size	The size of the stadd data buffer in bytes. The specified size must be greater than zero and less than or equal to 8128. The buffer must be within the task's address space.
stadd	The starting address of the data buffer. The address may be byte aligned.
type	Either 0 or 1 to indicate either a break or a long space.
tmo	An optional time-out count when used in conjunction with TF.TMO on read requests, IO.CON, and IO.ORG requests.

The time-out is specified as follows:

```
.BYTE x,y
```

where x is the number of ten-second intervals, up to 255., and y is the number of one-second interval, also up to 255. The longest possible time-out interval that can be specified is 255. seconds. If the time-out value is larger than 255 seconds, 255 seconds is used. Section 13.7 describes the effect of the time-out parameters on specific requests.

vfc — A character for vertical format control from Table 12-12 (Vertical Format Control Characters).

13.3.1 Device-Specific QIO Functions

Several of the device-specific functions described in this section can be issued by ORing a particular subfunction bit with another QIO function. These subfunction bits are specified in the following descriptions.

13.3.1.1 IO.ANS—The Answer function establishes a connection in answer mode, either in response to a ringing line, or if connection already exists. If a connection is not complete within 30 seconds, an IE.DNR error will be returned. The buffer address is required but is not used.

13.3.1.2 IO.ATA—IO.ATA is a variation of the Attach function. IO.ATA specifies an asynchronous system trap (AST) to process unsolicited events when called as follows:

```
QIO$C IO.ATA,...,<ast[,par2]>
```

When an unsolicited event occurs, the resulting AST serves as notification of the unsolicited event. Upon entry to the AST, the high byte of the top word on the stack contains par2, if it was specified. The low byte contains the event type. This word must be removed from the stack before exitting the AST. See section 13.6 for more information on unsolicited events.

IO.ATA is equivalent to IO.ATT ORed with the subfunction bit TF.AST.

13.3.1.3 IO.BRK—When issued, the IO.BRK function causes either a break or a long space to be sent. If parameter 1 is zero, a break is sent. If parameter 1 is one, a long space is sent.

On the XK communication port, a break will last for approximately 235 milliseconds, and a long space approximately 3.5 seconds.

13.3.1.4 IO.CON—The IO.CON function dials and connects a line in originate mode, as follows:

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

where stadd is the address of the telephone number to dial.

If TF.TMO is not specified, the request will complete when a connection is established or after 60 seconds.

13.3.1.5 SF.GMC—The Get Multiple Characteristics function returns driver characteristics information, as follows:

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

where stadd is the starting address of a data buffer of length "size" bytes. Each word in the buffer has the form:

```
.BYTE   characteristic-name
.BYTE   0
```

where characteristic-name is one the bit names given in Table 13-3. The value returned in the high byte of each byte-pair is value of that characteristic.

Table 13-3
XK Driver Characteristics for SF.GMC and SF.SMC Functions

Bit Name	*Valid Values*	*Meaning*
TC.ARC	0–9.	Auto-answer ring count (0 => don't answer)
TC.BIN	0,1	Enable or disable XON/XOFF support
TC.CTS	0,1	Resume or suspend output
TC.EPA	0,1	Odd or Even parity (if TC.PAR is specified)
TC.FSZ	Note 1	Character width including parity (if any)
TC.PAR	0,1 Note 1	Enable parity checking and generation
TC.RSP	Note 2	Receiver speed (bits-per-second)
TC.STB	1,2	Number of stop bits
TC.TBF	Note 3	Input ring buffer count or flush
TC.TRN	Note 4	Set translate table
TC.XMM	0,1	Disable or enable Maintenance mode
TC.XSP	Note 2	Transmitter speed (bits-per-second)
TC.8BC	0,1 Note 1	Pass 8-bit characters on input and output
XT.MTP	Note 5	Modem type

1. TC.FSZ is the frame size of a character. It is the number of data bits per character, plus 1 if parity is enabled.

   TC.FSZ and TC.PAR interact with each other to determine the number of data bits returned to the task. Table 13-4 shows the relationship of these characteristics.

   Two combinations do not appear in Table 13-4; TC.FSZ=9 with TC.PAR=0, and TC.FSZ=5 with TC.PAR=1. These two combinations are invalid, and the driver will return an error. To avoid this problem, always set the value of TC.FSZ first. The driver will automatically enable or disable parity if the value of TC.FSZ is 9 or 5.

   If the value of TC.FSZ is 8 or 9, the number of data bits returned to the task is further modified by the value of TC.8BC. If TC.8BC is set to 1, all 8 data bits will be returned to the task. If TC.8BC is set to 0, only 7 data bits will be returned to the task. Setting TC.BIN to a value of 1 or using the IO.RAL function will override the value of TC.8BC.

Table 13-4
TC.FSZ and TC.PAR Relationship

TC.FSZ	*TC.PAR*	*Number of Data Bits Returned to the Task*
9	1	8
8	0	8
8	1	7
7	0	7
7	1	6
6	0	6
6	1	5
5	0	5

2. TC.RSP and TC.XSP values and corresponding baud rates are:

Table 13-5
Receiver and Transmitter Speed Values (TC.RSP, TC.XSP)

TC.RSP or TC.XSP Value	*Actual Baud Rate (in bits-per-second)*
S.50	50
S.75	75
S.110	110
S.134	134.5
S.150	150
S.200	200
S.300	300
S.600	600
S.1200	1200
S.1800	1800
S.2000	2000
S.2400	2400
S.3600	3600
S.4800	4800
S.7200	7200
S.9600	9600
S.19.2	19200

3. The TC.TBF characteristic returns the number of unprocessed characters in the input buffer when used with SF.GMC. If there are more than 255 characters in the buffer, the value 255 will be returned. When used with SF.SMC, TC.TBF causes the input buffer to be flushed.

4. The translate table allows translation for either non-standard pulsing arrangements or for modems other than the DF03 which may be connected to the XK Communications port. The translate table is made up of three sections; a dial translation table, a start sequence string, and an end sequence string. Any or all sections of the translate table may be empty.

   The format of this characteristic is:

```
.BYTE    TC.TRN,count1,count2,count3
.BYTE    <dial_translate_table>
.BYTE    <start_sequence>
.BYTE    <end_sequence>
.EVEN
```

   where count1 is the length of the dial translate table, count2 is the length of the start sequence, and count3 is the length of the end sequence.

The dial translate table is a string of character pairs, input character followed by output character. This translate table is used to convert a telephone number, typically to remove format effectors such as "(", ")", "-", and " ". If a character in the telephone number matches a character in the input section of the dial translate table, the character is converted to the character from the output section. If the character from the output section is 0, the character from the telephone number will be ignored.

The start sequence string, if specified, will be sent to the autodialer before the phone number.

The end sequence string, if specified, will be sent to the autodialer after the phone number.

Note:

1. The next characteristic must begin on a word boundary.
2. This is a write-only parameter, and will return an SE.NIH error if used with the SF.GMC function.

5. The XT.MTP characteristic has the following values:

XTM.NO	No modem, hard-wired line
XTM.FS	USFSK - 0..300 baud Bell 103J
XTM.21	CCITTV.21 - 0..300 baud European
XTM.M1	CCITTV.23 Mode 1 - 75/0..300 split baud
XTM.M2	CCITTV.23 Mode 2 - 75/0..1200 split baud
XTM.PS	DPSK - 1200 baud Bell 212

13.3.1.6 IO.HNG—The IO.HNG function causes a line to be hung up.

13.3.1.7 IO.LTI—The IO.LTI function causes the driver to deliver unsolicited event notification AST's to a specified task, if the driver is not attached by another task. It is called as follows:

```
QIO$C IO.LTI,...,<stadd,size[,par3]>
```

where stadd is the address of a three word buffer of the form:

```
.WORD   AST_address
.RAD50  /first_half_of_task_name/
.RAD50  /second_half_of_task_name/
```

When an unsolicited event occurs, the resulting AST serves as notification of the unsolicited event. Upon entry to the AST, the high byte of the top word on the stack contains par3, if it was specified. The low byte contains the event type. This word must be removed from the stack before exitting the AST. See section 13.6 for more information on unsolicited events.

13.3.1.8 IO.ORG—The IO.ORG function initiates a connection in originate mode, assuming the line has already been connected. The buffer address is required but is not used.

13.3.1.9 IO.RAL—The Read All function causes the driver to pass all bits to the requesting task, when the value of TC.FSZ is 8 or 9. The driver does not mask out the high-order bit. This function is used to temporarily bypass the setting of the TC.8BC characteristic. Note that unlike the RSX-11M/M-PLUS terminal driver, this function does not pass CTRL/Q or CTRL/S to the requesting task. The TC.BIN must be set for these characters to be returned to the task.

IO.RAL is equivalent to IO.RLB ORed with the subfunction bit TF.RAL.

13.3.1.10 IO.RNE—The Read with No Echo function is accepted by the driver and the subfunction bit is ignored.

IO.RNE is equivalent to IO.RLB ORed with the subfunction bit TF.RNE.

13.3.1.11 SF.SMC—This function enables a task to set and reset the characteristics of the XK driver. Set Multiple Characteristics is the inverse function of SF.GMC. Like SF.GMC, is is called in the following way:

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

where stadd is the starting address of a data buffer of length "size" bytes. Each word in the buffer has the form:

```
.BYTE    characteristic-name
.BYTE    value
```

where characteristic-name is one of the bit names given in Table 13-3. The value returned in the high byte of each byte-pair is value of that characteristic.

13.3.1.12 IO.TRM—The IO.TRM function causes the driver to be unloaded. The task must be attached to issue this function. The driver will not unload unless the device is detached. This function requires a buffer address and count. The buffer is not modified.

13.3.1.13 IO.UTI—The IO.UTI function disables unsolicited event notification while the driver is not attached.

13.3.1.14 IO.WAL—The Write All function is accepted by the driver and the subfunction bit is ignored. The driver transmits all data bits in all cases.

IO.WAL is equivalent to IO.WLB ORed with the subfunction bit TF.WAL.

13.4 STATUS RETURNS

Table 13-6 lists error and status conditions that are returned by the communications driver to the I/O status block.

The SE.xxx codes are returned by the SF.GMC and SF.SMC functions as described in Sections 13.3.1.5 and 13.3.1.11. When any of these codes are returned, the low byte in the first word of the I/O status block will contain IE.ABO. The second IOSB word contains an offset (starting from 0) to the byte in error in the QIO's stadd buffer.

Table 13-6
XK Driver Status Returns

Code	*Reason*
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.PND	*I/O request pending* The operations specified in the QIO directive has not yet been executed. The I/O status block is filled with zeros.
IS.TMO	*Successful completion on a read* The input from the communications port was terminated by a time-out (non-zero value specified for the tmo parameter). The input buffer contains the bytes read.
IE.ABO	*Operation aborted* The specified I/O operation was cancelled 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.ALC	*Allocation failure* The total size of the phone number specified by an IO.CON request plus the start and end sequences was larger than the driver's internal buffer.
IE.CNR	*Connection rejected* Carrier was already present when an IO.CON request was issued.
IE.DAA	*Device already attached* The physical device-unit specified in the IO.ATT function wal 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, the subfunction bit has no effect.
IE.DNA	*Device not attached* The physical device-unit specified in an IO.DET or IO.TRM 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 that an attempt was made to perform a function on a line connected to a modem without carrier present, or to indicate that a connection was not established within the time-out period specified by an IO.CON, IO.ANS, or IO.ORG request.
IE.IFC	*Illegal function* A function code specified in an I/O request was illegal for the communications port.

Table 13-6 (Cont.)

Code	*Reason*
IE.OFL	*Device off-line* The physical device-unit associated with the LUN specified in the QIO directive was not online.
SE.NIH	*Characteristic not implemented* A characteristic other than those specified in Table 13-3 was named in an SF.GMC or SF.SMC request.
SE.VAL	*Illegal characteristic value* The new value specified in an SF.SMC request was not one of those listed in Table 13-3.

13.5 FULL-DUPLEX OPERATION

The XK driver attempts to simultaneously service one read request and one write request. Note that unlike the RSX-11M/M-PLUS full-duplex terminal driver, the SF.SMC function is NOT blocked until the line is idle. Resetting characteristics during I/O operations may cause unpredictable results.

13.6 UNSOLICITED EVENT PROCESSING

If a task attaches for unsolicited event AST's (IO.ATA), an AST will be dispatched whenever any of the events listed in Table 13-7 occur. When the AST is entered, the event type will be in the low byte of the top word of the stack, and par2 (IO.ATA) or par3 (IO.LTI) will be in the high byte. Note that the XTU.UI event is processed differently from the rest.

Table 13-7
Unsolicited Event Types

XTU.CD	Carrier detect
XTU.CL	Carrier loss
XTU.OF	XOFF received
XTU.ON	XON received
XTU.RI	Ring
XTU.UI	Unsolicited input

13.6.1 XTU.UI

If the event type is XTU.UI (unsolicited input), the AST becomes "disarmed" until a read request is issued by the task. Once the read request has completed, the AST is "armed" again for new unsolicited events.

13.7 TIME-OUT

The optional time-out parameter on read, IO.CON, and IO.ORG requests effects the action of the request. The following sections describe those effects.

13.7.1 Read Requests

tmo = 0	The request completes immediately after transferring as many characters as are available, less than or equal to the size parameter. The number of bytes transferred is returned in the second I/O status word.
tmo <> 0	The request completes after the time-out period or the requested number of bytes has been transferred. The number of bytes transferred will be returned in the second word of the I/O status block.

13.7.2 IO.CON

tmo = 0	The request completes immediately. If carrier is not present after 60 seconds, DTR and RTS are dropped.
tmo <> 0	The request completes after the time-out period, or after a connection is established. If the time-out period expires, DTR and RTS will be dropped, and an IE.DNR error will be returned in the first word of the I/O status block. If carrier comes up before the time-out period expires IS.SUC will be returned in the first word of the I/O status block.

13.7.3 IO.ORG

tmo = 0	The request completes immediately. If carrier is down, DTR and RTS will be dropped and an IE.DNR error will be returned in the first word of the I/O status block. If carrier is up, IS.SUC will be returned in the first word of the I/O status block.
tmo <> 0	The request completes after the time-out period, or after a connection is established. If the time-out period expires, DTR and RTS will be dropped, and an IE.DNR error will be returned in the first word of the I/O status block. If carrier is up (or comes up before the time-out period expires), IS.SUC will be returned in the first word of the I/O status block.

13.8 XON/XOFF SUPPORT

If XON/XOFF support is requested (TC.BIN = 0), the driver will transmit an XOFF whenever the ring buffer is three-quarters filled, and an XON whenever the buffer is then emptied below the one-quarter point. Because of this, tasks should not pass XON/XOFF control characters to the driver for transmission.

If an XOFF is received, transmission will be blocked. If the task is attached for unsolicited event AST's, an XTU.OF event will be dispatched. In any case, the TC.CTS parameter will reflect the XON/XOFF state of the line.

If XON/XOFF support is not requested (TC.BIN = 1), and the value of XT.MTP is XTM.NO (no modem), the Clear to Send line will be used in place of XON/XOFF control characters. State changes of this line will cause unsolicited event AST's, and modify the value of TC.CTS.

APPENDIX C
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. Sections are organized in the following sequence:

- ☐ I/O 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.

C.1 I/O STATUS CODES

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

C.1.1 I/O Status Error Codes

The octal number listed is the low-order byte of the complete word value (2's complement of the decimal number).

Name	*Decimal*	*Octal*	*Meaning*
IE.ABO	-15	361	Operation aborted
IE.ALN	-34	336	File already open
IE.BAD	-01	377	Bad parameter
IE.BBE	-56	310	Bad block
IE.BCC	-66	276	Block check error or framing error
IE.BLK	-20	354	Illegal block number
IE.BYT	-19	355	Byte-ligned buffer specified
IE.CNR	-73	267	Connection rejected
IE.CON	-22	352	UDC connect error
IE.DAA	-08	370	Device already attached
IE.DAO	-13	363	Data overrun
IE.DNA	-07	371	Device not attached
IE.DNR	-03	375	Device not ready
IE.DUN	-09	367	Device not attachable
IE.EOF	-10	366	End-of-file encountered
IE.EOT	-62	302	End-of-tape encountered
IE.EOV	-11	365	End-of-volume encountered
IE.FHE	-59	305	Fatal hardware error
IE.FLG	-89	247	Event flag already specified
IE.FLN	-81	257	ICS/ICR controller already offline
IE.IEF	-97	237	Invalid event flag
IE.IES	-82	256	Invalid escape sequence
IE.IFC	-2	376	Illegal function
IE.MOD	-21	353	Invalid UDC or ICS/ICR module
IE.NLK	-79	261	Task not linked to specified ICS/ICR interrupts
IE.NLN	-37	333	File not open
IE.NOD	-23	351	No dynamic memory available to allocate a secondary control block
IE.NST	-80	260	Task specified in ICS/ICR Link or Unlink request not installed

Name	*Decimal*	*Octal*	*Meaning*
IE.NTR	-87	251	Task not triggered
IE.OFL	-65	277	Device off line
IE.ONP	-05	373	Illegal subfunction
IE.OVR	-18	356	Illegal read overlay request
IE.PES	-83	255	Partial escape sequence
IE.PRI	-16	360	Privilege violation
IE.REJ	-88	250	Transfer rejected
IE.RSU	-17	357	Nonsharable resource in use
IE.SPC	-06	372	Illegal address space
IE.TMO	-74	266	Time-out error
IE.VER	-04	374	Unrecoverable error
IE.WCK	-86	252	Write check error
IE.WLK	-12	364	Write-locked device

C.1.2 I/O Status Success Codes

Decimal	*Octal*		
Name	*Bytes*	*Word*	*Meaning*
IS.CC	Byte 0: 1 Byte 1: 3	001401	Successful completion on read terminated by CTRL/C
IS.CR	Byte 0: 1 Byte 1: 15	006401	Successful completion with RETURN
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)

C.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.

C.2.1 Directive Error Codes

Name	*Decimal*	*Octal*	*Meaning*
IE.ADP	–98	177636	Invalid address
IE.IEF	–97	177637	Invalid event flag number
IE.ILU	–96	177640	Invalid logical unit number
IE.SDP	–99	177635	Invalid DIC number or DPB size
IE.ULN	–05	177773	Unassigned logical unit number
IE.UPN	–01	177777	Insufficient dynamic storage

C.2.2 Directive Success Codes

Name	*Decimal*	*Octal*	*Meaning*
IS.SUC	+01	000001	Directive accepted

C.3 I/O FUNCTION CODES

This section lists octal codes for all standard and device-dependent I/O functions.

C.3.1 Standard I/O Function Codes

Symbolic Name	*Word Equivalent*	*Octal Code (High Byte)*	*Octal 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

C.3.2 Specific Terminal I/O Function Codes

Slymbolic Name	*Word Equivalent*	*Octal Code (High Byte)*	*Octal 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
SF.GMC	002560	5	160	Get multiple characteristics
IO.GTS	002400	5	00	Get terminal support
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
SF.SMC	002440	5	40	Set multiple characteristics
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
IO.RSD	006030	14	30	Read special data
IO.WSD	005410	13	10	Write special data

C.3.3 Subfunction Bits

With IO.RLB, IO.RPR:

TF.RST	1
TF.BIN	2
TF.RAL	10
TF.RNE	20
TF.XOF	100
TF.TMO	200

With IO.WLB:

TF.WAL	10
TF.CCO	40
TF.WBT	100

With IO.ATT:

TF.AST	10
TF.ESQ	20

INDEX

P/OS System Reference Manual
Order No. AA-N620-TK
and Update Notice Number 1
Order No. AD-N620A-T1

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 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

Please cut along this line.

---- 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

Professional 300 Series Publications
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MASSACHUSETTS 01754

---- Do Not Tear - Fold Here ----