IBM RT PC Virtual Resource Manager Technical Reference Version 2.1

Virtual Resource Manager Device Support

Programming Family

IBM

Personal
Computer
Software

SC23-0817-0

Virtual Resource Manager Device Support

Programming Family

Personal Computer Software

First Edition (January 1987)

This edition applies to Version 2.1 of the Virtual Resource Manager, and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Changes are made periodically to the information herein; these changes will be incorporated in new editions of this publication.

IBM TECHNICAL NEWSLETTER

for the

RT Personal Computer Virtual Resource Manager

Virtual Resource Manager Device Support

—OVER—

Order Numbers:
08F3319
SN20-9883
September 25, 1987

08F3319
Printed in U.S.A.

Summary of Changes

This technical newsletter provides additional device support for the Small Computer Systems Interface device driver component of the Virtual Resource Manager.

A change to the text is indicated by a vertical bar to the left of the change.

Perform the following:

Remove Pages	Insert Update Pages
5-95 and 5-96	5-95 and 5-96
5-101 to 5-106	5-101 to 5-106.2
5-113 and 5-114	5-113 and 5-114

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM TECHNICAL NEWSLETTER

for the

RT Personal Computer Virtual Resource Manager

Virtual Resource Manager Device Support

—OVER—

Order Numbers:
79X3823
SN20-9859
June 26, 1987

TB79X3824
Printed in U.S.A.

Summary of Changes

This technical newsletter provides additional command and device support available with the Virtual Resource Manager.

A change to the text is indicated by a vertical bar to the left of the change.

Perform the following:

Remove Pages	Insert Update Pages
vii and viii	vii and viii
2-3 and 2-4	2-3 and 2-4
3-9 and 3-10	3-9 and 3-10
3-191 to 3-198	3-191 to 3-198
4-1 to 4-4	4-1 to 4-4
4-43 and 4-44	4-43 and 4-44
4-51 to 4-56	4-51 to 4-56
4-59 to 4-68	4-59 to 4-68
4-89 and 4-90	4-89 to 4-102
5-93 and 5-94	5-93 and 5-94
5-103 to 5-114	5-103 to 5-114

Note: Please file this cover letter at the back of the manual to provide a record of changes.

About This Book

Audience and Purpose

This book describes the device support provided by the VRM, and focuses on the device management role the VRM plays in the RT Personal Computer[1]. Topics discussed include the initial program load of the VRM, coding considerations for developers who wish to add device drivers or managers to the system, definition of the virtual terminal and block I/O subsystems, and the programming interfaces to the predefined VRM device drivers.

Virtual Resource Manager Programming Reference describes the VRM programming environment.

This book is intended for systems programmers and developers who need to understand the device support provided by the VRM for the RT PC. The reader of this book is expected to have an understanding of hardware and operating systems fundamentals.

How to Use This Book

This book describes the VRM from its own IPL and configuration to its interfaces with its principal subsystems, the minidisk manager, and predefined device drivers. This book also describes the coding considerations and other requirements that must be met to add code to the VRM.

The book consists of the following sections:

Chapter 1, "Virtual Resource Manager Initial Program Load" on page 1-1 describes how the VRM and its supported devices are IPLed and configured. This chapter also describes how certain code modules (such as device driver modules) are loaded at IPL time. In addition, this chapter tells how to IPL the system with a nonbase display or add code to the VRM minidisk.

Chapter 2, "VRM Device Driver Concepts" on page 2-1 explains how device drivers work in the VRM. In addition to defining the common routine interface to device drivers and managers, this section provides the information necessary to develop and install code into

---

1 RT Personal Computer, RT PC, and RT are trademarks of International Business Machines Corporation.

the VRM. This includes instructions for converting **a.out** format object modules to VRM-executable object modules.

Chapter 3, "Virtual Terminal Subsystem" on page 3-1 describes the virtual terminal subsystem interface to the virtual machine. The characteristics of the virtual terminal subsystem are defined, as well as the commands and interrupts that cross the VMI.

Chapter 4, "Block I/O Subsystem" on page 4-1 describes how block I/O devices function in the VRM and how they interface with the operating system. This chapter includes a description of a generic block I/O device driver, the block I/O device manager, and some specific block I/O device drivers.

Chapter 5, "IBM Predefined Device Drivers" on page 5-1 defines the programming interfaces to the native VRM device drivers. This section includes information on fixed disks, diskettes, printers, streaming tapes, and other device types.

A glossary and index follow these chapters and appendices.

A Reader's Comment Form and Book Evaluation Form are provided at the back of this book. Use the Reader's Comment Form at any time to give IBM information that may improve the book. After you become familiar with the book, use the Book Evaluation Form to give IBM specific feedback about the book.

Please note the following items regarding hexadecimal notation, reserved fields, the bit padding and numbering conventions, and highlighting conventions used in this book:

- A hexadecimal value as expressed in this publication is preceded by a zero and a lowercase ***x***. For example, the hexadecimal value 'F3' is represented as 0xF3.
- The value of reserved fields input to the Virtual Resource Manager must be set equal to zero. The value of reserved fields returned by the VRM is unpredictable.
- The bit-numbering convention used by IBM has the most significant bit on the left and the least significant bit for a given field on the right. For example, in a 32-bit word, the most significant bit (bit 0) is the leftmost bit and the least significant bit (bit 31) is the rightmost bit.
- High-order bits in a given field not used to express a value are padded with zeroes. For example, if an input parameter sent in a 32-bit register is only a 16-bit value, the value occupies bits 16 through 31 of the register and bits 0 through 15 of the register are padded with zeroes.
- Terms highlighted in bold-faced italics (such as ***example***) are specific to the RT PC and are included in the glossary. Terms highlighted in bold type (such as **example**) are system-generated items, such as commands, file names, and so on. Terms or phrases that appear in monospace type (such as `example`) are examples of what you might see on a display screen or what you must enter into the system to perform a given function.

Related Information

The following RT PC publications provide additional information on topics related to the VRM. Depending on the tasks you want to perform and your experience level, you may want to refer to the following publications:

- *IBM RT PC Installing and Customizing the AIX Operating System* provides step-by-step instructions for installing and customizing the Advanced Interactive Executive[2] Operating System, including how to add or delete devices from the system and how to define device characteristics. This book also explains how to create, delete, or change AIX and non-AIX minidisks.
- *IBM RT PC Installing the Virtual Resource Manager* provides step-by-step instructions for installing the Virtual Resource Manager and shows you how to change the IBM-recommended choices to suit your system needs. (Available as a separate volume only when the Virtual Resource Manager is purchased separately from the AIX Operating System.)
- *IBM RT PC Hardware Technical Reference* is a three-volume set. Volume I describes how the system unit operates, including I/O interfaces, serial ports, memory interfaces, and CPU interface instructions. Volumes II and III describe adapter interfaces for optional devices and communications and include information about IBM Personal Computer family options and the adapters supported by 6151 and 6150. (Available optionally)
- *IBM RT PC Assembler Language Reference* describes the IBM RT PC Assembler Language and the 032 Microprocessor and includes descriptions of syntax and semantics, machine instructions, and pseudo-operations. This book also shows how to link and run Assembler Language programs, including linking to programs written in C language. (Available optionally)
- *IBM RT PC AIX Operating System Programming Tools and Interfaces* describes the programming environment of the AIX Operating System and includes information about using the operating system tools to develop, compile, and debug programs. In addition, this book describes the operating system services and how to take advantage of them in a program. This book also includes a diskette that includes programming examples, written in C language, to illustrate using system calls and subroutines in short, working programs. (Available optionally)
- *IBM RT PC Messages Reference* lists messages displayed by the IBM RT PC and explains how to respond to the messages.
- *IBM RT PC AIX Operating System Commands Reference* lists and describes the AIX Operating System commands.

---

2 Advanced Interactive Executive and AIX are trademarks of International Business Machines Corporation.

- *IBM RT PC C Language Guide and Reference* provides guide information for writing, compiling, and running C language programs and includes reference information about C language data structures, operators, expressions, and statements. (Available optionally)
- *IBM RT PC Problem Determination Guide* provides instructions for running diagnostic routines to locate and identify hardware problems. A problem determination guide for software and three high-capacity (1.2MB) diskettes containing the IBM RT PC diagnostic routines are included.
- *IBM RT PC Keyboard Description and Character Reference* describes the national character and keyboard support for the 101-key, 102-key, and 106-key keyboards, including keyboard position codes, keyboard states, control code points, code sequence processing, and nonspacing character sequences.
- *RT PC VRM/Hardware Quick Reference* contains brief descriptions of the hardware and Virtual Resource Manager. This booklet includes information on command parameters and return codes and hardware and memory layout data.
- *IBM RT PC AIX Operating System Technical Reference* describes the system calls and subroutines that a C programmer uses to write programs for the AIX Operating System. This book also includes information about the AIX file system, special files, file formats, GSL subroutines, and writing device drivers. (Available optionally)

Ordering Additional Copies of This Book

To order additional copies of this book (without program diskettes), use either of the following sources:

- To order from your IBM representative, use Order Number SBOF-0136.
- To order from your IBM dealer, use Part Number 79X3822.

Two binders and the *Virtual Resource Manager Technical Reference* are included with the order. For information on ordering a binder, books, or the *RT PC VRM/Hardware Quick Reference* separately, contact your IBM representative or your IBM dealer.

Contents

Figures

Chapter 1. Virtual Resource Manager Initial Program Load

Contents

About This Chapter

This chapter describes the sequence of events associated with the installation of and initial program load (IPL) of the VRM. In addition, this chapter describes the devices configured at VRM and operating system configuration and the permission bits that determine whether different modules are loaded, executed, and/or deleted. Also discussed is IPL with non-base devices and the strategy for revising or reinstalling licensed program products that reside on the VRM minidisk.

VRM IPL and Configuration

The VRM, because of its role in controlling system hardware, is loaded before the operating system code. VRM code is loaded in stages and can be done from a diskette or from a minidisk on the fixed disk. A minidisk is a logical partition of a fixed disk. The VRM code on both the minidisk or diskette is in the form of an AIX file system.

When you first turn on the machine, read-only memory (ROM) searches the possible IPL devices for an IPL record. The diskette IPL device is always searched first. Therefore, when a diskette containing an IPL record is inserted into a diskette drive, this diskette will be the IPL source (even if a VRM minidisk already exists). Also resident in ROM and invoked when the machine is turned on are several ***power-on self tests*** (POST). A POST performs a presence check and verifies that the hardware is basically functional. POSTs are run on the main processor card, any random-access memory cards, the system planar hardware (such as the keyboard) and the fixed disk and diskette adapters.

In addition to the ROM-resident POSTs, another set of POSTs reside in a configurable directory on the VRM minidisk and VRM diskette. These loadable POSTs perform tests similar to those performed by the ROM-resident tests. Loadable POSTs can also supply some additional functions, such as passing a define device structure to the VRM.

Both ROM-resident and loadable POSTs write the results of their tests to a control block called the POST control block (PCB). The PCB is located in virtual equals real memory at address 0x800 of RAM. The PCB format is shown in Figure 1-1 on page 1-5. PCB entries from 0x1000 to 0x1800 are written over when the VRM is installed, but the information is preserved on the backup copy of the PCB that is stored on the fixed disk.

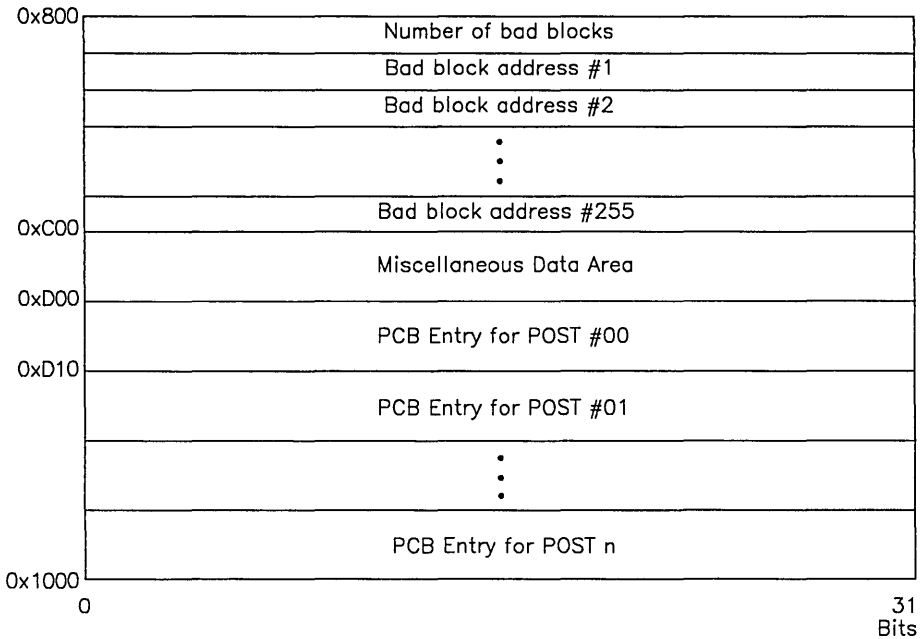


Figure 1-1. POST Control Block Layout

Data contained in the PCB includes:

- A list of bad blocks on 128K-byte boundaries
- Miscellaneous data
- A four-word entry for each POST. The format of a PCB entry is shown in Figure 1-2.

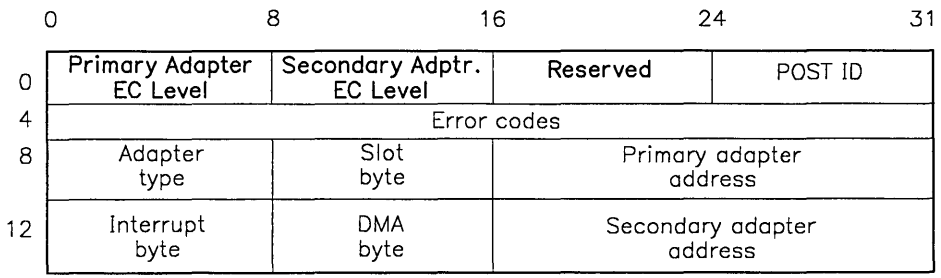


Figure 1-2. PCB Entry

The fields in a PCB entry are defined as follows:

- POST ID

  This byte identifies the offset into the PCB starting at 0x0D00.

- Error codes

  This word consists of two status bits and 30 bits of error information that are unique by adapter. Bit 0 indicates good/bad status and bit 1 specifies adapter presence or absence.

- Adapter type

  This byte provides a two-digit value that identifies which IBM-supplied adapter was tested. The following values are defined:

  19 RT PC Floating-Point feature (Floating-Point Accelerator, Advanced Floating-Point Accelerator, or APC Floating-Point Coprocessor)
  1A RT PC 6150 Native RS-232C Serial Adapter
  20 RT PC Personal Computer AT® Coprocessor Option
  23 RS-232C Serial portion of the AT® Serial/Parallel Adapter
  28 RT PC Multiprotocol Adapter
  2A Parallel portion of the AT Serial/Parallel Adapter or the IBM Monochrome Display and Printer Adapter
  30 RT PC Token Ring Network Adapter
  31 IBM PC Network Adapter
  32 RT PC 5080 Attachment Adapter
  33 RT PC 3278/79 Advanced Emulation Adapter
  35 RT PC 4-Port Asynchronous RS-232C Adapter
  38 RT PC 5080 Peripheral Adapter
  39 RT PC 4-Port Asynchronous RS-422A Adapter
  3A RT PC Secondary Serial Link Adapter
  40 RT PC Baseband Adapter
  41 IBM PC Enhanced Graphics Display Adapter
  43 RT PC Advanced Monochrome Graphics Display Adapter
  45 RT PC Advanced Color Graphics Display Adapter
  47 RT PC Extended Monochrome Graphics Display Adapter
  49 IBM Monochrome Display and Printer Adapter
  4A RT PC IBM 5081 Color Display Adapter 1
  4B RT PC IBM 5081 Color Display Adapter 2
  52 Fixed-disk portion, Personal Computer AT Fixed Disk and Diskette Drive Adapter
  53 Diskette portion, Personal Computer AT Fixed Disk and Diskette Drive Adapter
  5B RT PC Small Computer Systems Interface Adapter for IBM 9332 DASD Attachment
  78 RT PC Keyboard
  79 RT PC Mouse
  7A IBM 5080 Dials Feature
  7B IBM 5080 Lighted Program Function Keyboard Feature
  80 IBM 5083 Tablet
  82 RT PC ESDI Magnetic Media Adapter

83 RT PC Sound Device
85 RT PC Streaming Tape Drive Adapter
86 RT PC High-Function ESDI Magnetic Media Adapter
EE IBM Personal Computer AT Bus-Attached RAM for the Coprocessor Option.

- Slot byte

  This value indicates the slot number of the adapter. The first four bits specify the slot number of the secondary adapter and the second four bits specify the slot number of the primary adapter. A value of 0xFF indicates that no slot number was found (adapter is absent).

- Primary adapter address

  This halfword indicates the primary adapter's I/O bus address (if the adapter has an I/O bus address).

- Interrupt byte

  This byte contains the interrupt level used by the primary and secondary adapters. The first four bits indicate the interrupt level of the secondary adapter and the next four bits indicate the interrupt level of the primary adapter. A value of 0x88 indicates the adapter uses no interrupts or is not present.

- DMA byte

  This byte contains the direct memory access (DMA) levels used by the primary and secondary adapters. The first four bits represent the DMA level used by the secondary adapter and the second four bits represent the DMA level used by the primary adapter. A value of 0xFF indicates that no DMA level is used.

- Secondary adapter address

  This halfword specifies the secondary adapter's I/O bus address (if the adapter has an I/O bus address).

After the ROM-resident POSTs execute, ROM must locate the IPL record, which is found at cylinder 0, head 0, sector 1 of the fixed disk or diskette. The EBCDIC character string 'IBMA' is contained in the first word of the IPL record. ROM reads the IPL record and then reads the loadlist processor (LLP) into memory.

The LLP is a standalone program that runs directly on the hardware. The LLP reads the AIX file system and sequentially processes the files and subdirectories in the loadlist (**ldlist**) directory. Figure 1-3 shows the directory structure of the VRM minidisk, and Figure 1-4 on page 1-8 shows the directory structure on the VRM diskette.

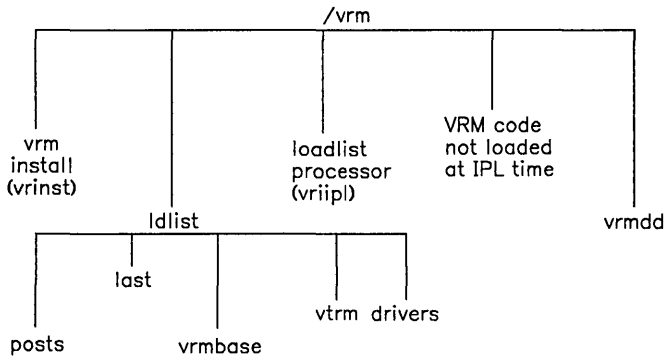


Figure 1-3. Directory Structure of the VRM Minidisk

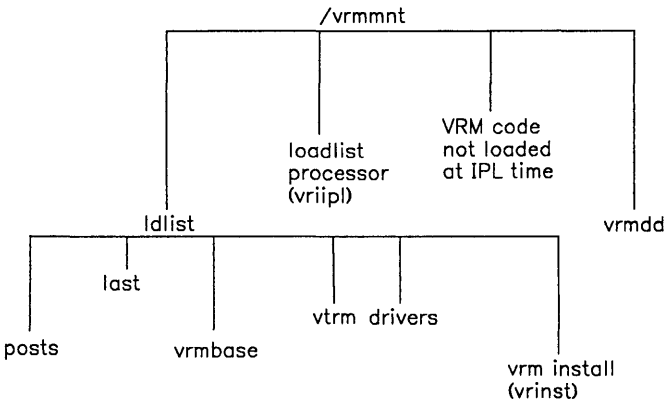


Figure 1-4. Directory Structure of the VRM Install Diskette

Permission Bit Conventions

Each of the files in the loadlist have permission bits that determine whether the loadlist processor loads, executes, and/or deletes the file. The LLP translates permission bits for files as follows:

`r-- --- ---` Load the module

`--x --- ---` Execute the module

`-w- --- ---` Delete the module.

The LLP skips entries with the following permission bits:

- Files

  -wx --- --- (300)
  -w- --- --- (200) Only the first 3 bits are significant.
  --x --- --- (100)

- Directories

  rw- --- --- (600)
  r-- --- --- (400) Only the first 3 bits are significant.
  -w- --- --- (200)

The VRM translates permission bits for files as follows:

--- r-- --- Bind the module to itself

--- --x --- Execute the module

--- -w- --- Unused.

For directories, the LLP translates permission bits as follows:

--x --- --- Search the directory.

Valid combinations of permission bits are listed and defined below.

- rwx --- --- (700)

  The LLP loads, executes, and deletes these modules. The module must conform to the following criteria:

  - The module must have a TOC header.
  - The module must be memory-location independent, at origin 0x60.
  - Relocation symbols must be relative to section 1 or 2 only.
  - The module cannot use any operating system routines.
  - The module must run on the stack of the LLP.
  - The module cannot have unresolved external references.
  - The module is loaded in high memory to execute.

- r-- r-- --- (440)

  The LLP loads these modules. The module must conform to the following criteria:

  - The module must have a TOC header.
  - No binding or relocating of the module is done by the LLP.
  - The module is loaded in low memory.

- --- r-- --- (040)

  If the IPL device is diskette, the LLP loads the module the same way it loads r-- r-- --- modules. If the IPL device is fixed disk, the LLP does not load the module. Instead, the module is

mapped. A mapped module's location on the fixed disk is entered into a table that is passed to the VRM nucleus module.

- `r-- r-x ---` (450)

  This is the VRM nucleus module.

  The LLP transfers control to this module after all loadlist directory entries are processed. The module must conform to the following criteria:

  - The module must have a TOC header.
  - The LLP does no relocating of the module.
  - A prefetch of the TOC header is done to determine the origin. If the origin is greater than 0x60, then:
    - The TOC header is stripped off before the module receives control.
    - An absolute memory address is allocated.
    - If the origin is 0x1000, half of the 4K POST control block is deallocated.

- `r-- --x ---` (410)

  The LLP recognizes this module as a virtual machine. The following loading requirements are necessary:

  - Instead of a TOC header, the module has a 512-byte virtual machine IPL record. See *VRM Programming Reference* for more information on virtual machine IPL records.
  - The module must start on a 2K-byte boundary.
  - The module size is considered the file size.
  - The LLP can load only one virtual machine and it must be the last module in the loadlist directory. The VRM diskette contains a virtual machine (vrm install) that creates the page space and VRM minidisks. This virtual machine then loads the VRM code onto the VRM minidisk.

Standard entries in the loadlist directory include:

- POSTs (**posts**)

  Loadable POSTs test optional features such as communications adapters, printers, additional displays, storage adapters, and so on. Loadable POSTs in general perform the same kind of tests as those performed by the ROM-resident POSTs. The posts directory allows for extension and customization of the system.

- Last loadable POST (**llasth.0000.00**)

  The last POST is a file that is maintained outside and immediately following the POST directory. Because this file indicates no more POSTs are to be processed, the POST directory can be modified without dependencies. This file writes the information contained in the PCB to fixed-disk sectors 34 through 49.

- VRM code (**vrmbase**)

  This directory contains the base VRM code, including the fixed disk device driver and VRM debugger.

- Virtual Terminal Resource Manager (**vtrm**)

  The VTRM is a device manager responsible for the terminal subsystem devices: display, keyboard, locator, and speaker. The code within this directory determines the terminal subsystem configuration including type of display hardware, necessary physical and virtual device drivers, fonts, and presence of locator hardware.

- Other device drivers (**drivers**)

  This directory contains some of the remaining device drivers, including diskette and streaming tape.

- Install virtual machine (**vrinst.0050.00**)

  The file system on the VRM diskette includes an install virtual machine. When the VRM is loaded from diskette, this file is IPLed as a virtual machine. The install virtual machine sets up the fixed disk with VRM and page space minidisks. When these minidisks are established, the operating system can be installed and then IPLed. Note that only one virtual machine can be loaded from diskette and that the virtual machine module must be the last file in the loadlist directory.

Other files that are neither located under the loadlist directory nor loaded or mapped at IPL time include:

- Device drivers configured from the operating system (**vrmdd**)

  This file contains device drivers for optional devices that are shipped with the VRM but defined and initialized when the operating system is IPLed. The VRM device drivers for printers and asynchronous devices are located in this directory.

- Install virtual machine (**vrinst.0050.00**)

  This file is located under the **/vrm** directory on the VRM minidisk, as opposed to being located under the loadlist directory on the diskette. See Figure 1-3 on page 1-8 and Figure 1-4 on page 1-8 for the relative locations of this file on the VRM minidisk and diskette.

Files processed by the loadlist processor must conform to the following naming convention:

```
xxxxxx.yyyy.zz
```

Each module name is broken down into the following fields:

xxxxxx This is the module's unique identifier. Note that it **must** be six characters in length. Valid characters include any alphanumeric character and any symbol with the exception of *, ?, [and].

yyyy This field is a hexadecimal value that represents the module's IOCN. This value must be in the range 0x0001 to 0x0400 if the module has an assigned IOCN. If the module has no IOCN (such as power-on self tests), this value is set equal to zero. This field **must** be four hexadecimal digits in length.

zz This field is the module's match code.

Match codes are used to tell the loadlist processor whether to load a particular module. When the loadlist processor executes a module (permission bits = 700), the module can return a list of one or more match values to the loadlist processor. For any succeeding modules in the same directory, the LLP processes the module only if its filename contains a match value of '00' or a value already returned to the LLP by another module. The LLP skips the remaining modules in that directory.

Match values returned from a module are kept only for the directory in which the module resides. When the LLP completes processing all the modules in a given directory, it discards all the match values returned from that directory's modules.

All modules executed by the loadlist processor must conform to the following register conventions. Note that GPRs 2 through 4 are input parameters and GPRs 5 and 6 are return parameters. Both GPR1 and GPR14 contain the address of the stack. The contents of GPRs 6 through 15 must not be changed.

GPR2 = Contains the address of the POST control block
GPR3 = Contains the address of the ROM entry point table
GPR4 = Contains the address of the define device area (DDA)
GPR5 = Contains the 32-bit address of a pointer to the list of match values returned by the module

A called routine that does not pass a list of match values to the loadlist processor should store a value of -1 (0xFFFFFFFF) in the address pointed to by the value in GPR5. Figure 1-5 shows the format of an entry in the match list.

GPR6 = This register is stored on the stack and contains the 32-bit address of a pointer to an entry to be added to the define device area.

A called routine that does not pass an entry for the DDA should store a -1 in the address pointed to by the value in GPR6.

On return, GPR2 contains 0 for successful completion. Values not equal to zero indicate unsuccessful completion.

If any non-zero return codes (unsuccessful processing) are returned to the loadlist processor, the code '23' is displayed for 10 seconds on the light-emitting diodes (LEDs) to reflect this fact.

Each match list entry (as shown in Figure 1-5) is an 8-byte field. The first two bytes are reserved, the next two bytes are the 2-character match code, and the next 4 bytes point to the next entry in the list. The last entry in the list has a value of -1 to indicate the last entry in the list.

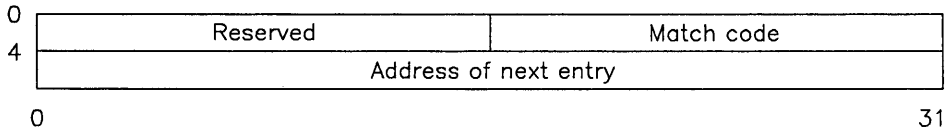


Figure 1-5. Format of a Match List Entry

The define device area (DDA) is an area built by the loadlist processor that contains the define device structures for certain devices. Figure 1-6 shows the format of a typical entry in the DDA.

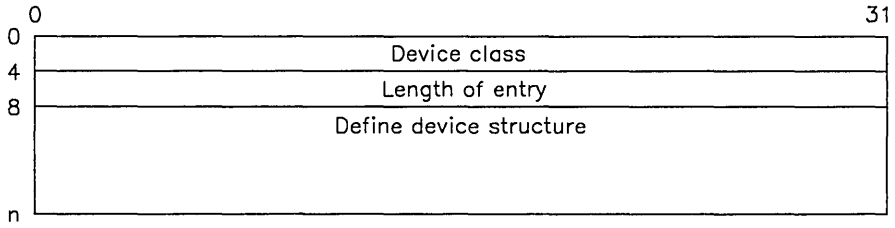


Figure 1-6. Define Device Area Entry

The fields in each DDA entry passed back from a called module are defined as follows:

- Device class (four bytes)

  Valid values for this field include:

  1 = This indicates the device is a fixed disk.
  2 = This indicates the device is a diskette.
  3 = This indicates the device is a keyboard.
  4 = This indicates the device is a display.
  5 = This indicates the device is the virtual terminal resource manager.
  6 = This indicates the device is the VRM debugger.
  7 = This indicates the device is a streaming tape.
  8 = This indicates the device is a locator.
  9 = This indicates the device is a speaker.

- Length of entry (four bytes)

  This field indicates the length of the entry in bytes. This length is simply the length of the DDS plus eight.

- DDS

  This field contains the device's DDS, including the DDS header, device characteristics, hardware characteristics, and error log fields (if applicable). For more information on the specific fields in a DDS, see *VRM Programming Reference.* The DDSs for some IBM-supplied devices are described in Chapter 5, "IBM Predefined Device Drivers" on page 5-1.

The LEDs are the only way for you to know what is happening in the system during IPL. In addition to the '23' return code that displays on the LEDs for unsuccessful loadlist processing, other codes you might see on the LEDs and their meanings are defined on the following page.

02 = Read error
03 = Insufficient space to load the module
04 = Insufficient space for the DDA
05 = Insufficient space for a data structure
06 = Insufficient space for a data structure
07 = Insufficient space for the match list
08 = Relocation symbol type not supported
09 = File type not supported
10 = Invalid device
11 = Invalid index for address
12 = Inode out of range
13 = Address out of range
14 = File not found
15 = No freeblock header space
16 = Freeing freespace
17 = Address already allocated
18 = Module not on a word boundary
19 = Invalid file name
20 = Invalid IOCN in the file name
21 = Invalid IOCN in the DDS
22 = Invalid length field in the DDS
23 = Execution of loaded program failed
24 = No entry point found
25 = Bad load module
45 = Base devices missing
46 = Probable programming error
47 = Error in **vtcp** command
48 = **Define Code SVC** failed for a module on the **vtcp** command line
49 = Insufficient memory for segment to read in the code
50 = **Define Device SVC** failed for a control program
51 = **Attach Device SVC** failed for a control program
52 = **Define Device SVC** failed for a new device
53 = **Define Device SVC** failed for the VTRM
54 = Unsuccessful initialization of the VTRM
55 = Diskette I/O error during VRM install
99 = Machine or program check error.

After all the entries in the loadlist are processed, the LEDs are cleared and control passes to the VRM nucleus module (permission bits = 450).

VRM IPL/Install with Non-Base Devices

The preceding IPL discussion describes how the VRM is installed and how devices are configured when using IBM code and hardware. The RT PC system, however, can be expanded with non-IBM code and devices. "Installing Code from a Virtual Machine" on page 2-29 describes how to add code to an IPLed system from a virtual machine. Non-base devices required by the VRM install program, such as displays, are added by VRM install in this way. The code for such non-base devices must be installed by way of a separate diskette during the VRM install process. *IBM RT PC Installing and Customizing the AIX Operating System* describes the steps for installing the VRM and IPLing the system with non-base devices and code. "Display Device Driver Considerations" on page 3-140 describes how to develop a device driver for a non-base display. The following section describes:

- The commands used to make permanent or temporary changes to the install process.
- The media (diskette) and formatting requirements for the new code.
- The update strategy for revised code.

Commands to Change the Install Process

The following commands can be used to make temporary or permanent changes in the RT PC IPL/install process:

- **addf** – add a file to the VRM minidisk
- **chmd** – change the permission bits of a file on the VRM minidisk
- **delf** – delete a file from the VRM minidisk
- **vtcp** – add non-base display code at VRM install time.

These commands can be placed in a control file on a 1.2M byte or 360K byte AIX file system diskette. The control file must be named **Inst.Batch** and must reside on the root of the diskette file system. The first command begins with the first character in **Inst.Batch**. Each command terminates with a newline character (0x0A) and the next command (if any) begins immediately after the newline character. Commands may be up to 128 characters in length (including the newline character). An asterisk (*) in column 1 indicates a comment record.

The VRM install program will read and execute the control file when the diskette is inserted during VRM install. *IBM RT PC Installing and Customizing the AIX Operating System* describes when and how to insert the device diskette during the VRM install process.

Of the preceding commands, **vtcp** makes a temporary change to the running copy of the VRM to allow a non-base display to be used by the VRM install program. This is considered a temporary change because the code is not present for subsequent IPLs. When the VRM minidisk is changed with **addf**, **delf**, or **chmd**, the changes are considered permanent because they will be in effect for subsequent IPLs. Of course, you can add non-base display code to the system with **vtcp** and then add the code to the VRM minidisk with **addf**. The non-base display code will then be used for subsequent IPLs.

These commands are defined on the following pages. Each definition includes the command syntax, input parameters, and possible error conditions.

addf — Add a file to the VRM minidisk

Description: The **addf** command reads the specified input file from a diskette and writes the output file to the same path on the VRM minidisk. Subdirectories indicated by the path name that do not exist on the fixed disk file system are created by this command. For this reason, extreme care must be used when naming files to be added to the VRM minidisk.

This command distinguishes between files added to the minidisk for the first time and files that are being updated or reinstalled. Code added to the VRM minidisk falls into three categories. They are:

- Code added for a program product that puts no history information on its diskette.
- Code added for a program product that does put a history file on its diskette.
- Code added to update an existing program product with a history file on the VRM minidisk. These diskettes should contain an **updlevel** file for the program product.

If code is added from a diskette with no history information, or if a program product is being installed for the first time, naming conflicts result in errors.

If a program product is being updated or reinstalled, a naming conflict causes the file to be deleted and replaced. See "Revising Code in the VRM" on page 1-24 for more details on history files.

Files added to the VRM minidisk are given the same permission bits as the source files on the diskette. Files for display device drivers, device-specific modules, and fonts should have permission set to 040 octal. See "Permission Bit Conventions" on page 1-8 for more information on permissions.

Command format:

```
addf <diskette_name> [insert_name]
```

The input parameters are defined as follows:

- `diskette_name` – indicates the full path name of the file to add from the diskette
- `insert_name` – is an optional parameter that specifies the simple file name of a file after which the added file is to be placed

  This parameter allows you to place a file in a specific location in a directory and is sometimes necessary for order-dependent modules. The specified file will be inserted in the same directory as `diskette_name`. To designate a file as the first in a directory, insert it after the dot dot (..) entry.

  If the insert option is used but the specified file is not found on the VRM minidisk, the added file is placed in the first available directory entry (usually at the end of the directory). No error is given.

Errors:

Errors that occur with this command cause a message with the prefix 046 to appear on the display. These messages are described in *IBM RT PC Messages Reference.*

The following error conditions may occur with **addf**:

- The VRM file system is full (no free space).
- The VRM file system contains the maximum number of files.
- The VRM file system is corrupt.
- An invalid argument, too few arguments, or too many arguments were found on the command line of the **Inst.Batch** file.
- An I/O error may result from use of a bad diskette.
- A file referenced on the command line was not found on the diskette.
- The specified file already exists on the VRM minidisk and no history file is present.

chmd — Change permission bits of a file on the VRM minidisk

Description: The **chmd** command allows you to change the permission bits for a file on the VRM minidisk. "Permission Bit Conventions" on page 1-8 explains how permission bits cause a module to be loaded, executed, ignored, and so on.

Command format:

```
chmd <disk_name> <permission>
```

The input parameters are defined as follows:

- `disk_name` – indicates the full path name of the file on the VRM minidisk for which permission bits will be changed
- `permission` – indicates the new permission to assign to the file

  This parameter consists of three octal digits representing the nine permission bits for a file.

Errors:

Errors that occur with this command cause a message with the prefix 046 to appear on the display. These messages are described in *IBM RT PC Messages Reference.*

The following error conditions may occur with **chmd**:

- An invalid argument, too few arguments, or too many arguments were found on the command line of the **Inst.Batch** file.
- The input permission value is invalid.
- An I/O error may result from use of a bad diskette.
- The file referenced on the command line was not found on the VRM minidisk.

delf — Delete a file from the VRM minidisk

Description: The **delf** command allows you to remove a file from the VRM minidisk. If the supplied file name is a directory, the directory is not deleted unless it is empty. This function should be used with extreme caution. If the wrong files are deleted from the VRM minidisk, system performance may be severely compromised.

Command format:

```
delf <disk_name>
```

The input parameter is defined as follows:

- `disk_name` — indicates the full path name of the file on the VRM minidisk to delete.

Errors:

Errors that occur with this command cause a message with the prefix 046 to appear on the display. These messages are described in *IBM RT PC Messages Reference.*

The following error conditions may occur with **delf**:

- An I/O error may result from use of a bad diskette.
- A file referenced on the command line was not found on the VRM minidisk.
- An invalid argument, too few arguments, or too many arguments were found on the command line of the **Inst.Batch** file.

vtcp — Add non-base display code at VRM install time

Description: The **vtcp** command allows you to add non-base display code at VRM install time. Examples of non-base display code include a device-specific module of the VTMP, a display device driver, a new font, and so on.

This command is necessary when the VRM install program cannot use the IBM-supplied VTRM. The **vtcp** command requires a control program to inform VRM install of changes to the VTRM DDS and to supply a DDS for the non-base display being added. The control program itself has a DDS that includes a copy of the VTRM DDS and the DDS of the non-base device. Figure 1-7 on page 1-22 shows the control program DDS. For a description of the IBM-supplied VTRM DDS, see "Virtual Terminal Resource Manager" on page 3-202.

Command format:

```
vtcp <dds_size> <control_program> <device_driver> [more_code] . . .
```

The input parameters are defined as follows:

- `dds_size` – a decimal number that indicates, in bytes, the size of the non-base display device driver DDS
- `control_program` – indicates the full path name of the control program

  The control program revises certain fields in the VTRM DDS so that new device drivers (and other code modules) are reflected there. The control program also supplies the DDS for the non-base display. Because the control program is defined to the system as a device driver, it must handle the following device driver entry points:

  - Define device (0x3C)
  - Initialize device (0x3D)
  - Initiate I/O (0x1E)
  - Terminate (0x3E)

  Other entry points, such as check parameters, are not required.
- `device_driver` – indicates the full path name of the device driver code to be added
- `more_code` – indicates the full path names of additional modules to be added

  This parameter is necessary for any modules that must be configured in the system with a **Define Code SVC**, such as a font or a device-specific module of the VTMP. See "Display Device Driver Considerations" on page 3-140 for details on this type of module.

The VRM install program reads the **vtcp** command from diskette and then assigns an IOCN and issues a **Define Code SVC** for each module specified on the command line.

VRM install then builds a DDS for the control program, assigns it an IODN, and issues a **Define Device SVC** for it. At this point, the control program receives control at its define device entry point and has addressability to its DDS. The control program must then update the copied VTRM DDS (see Figure 1-7 on page 1-22) and supply a DDS for the non-base device being added to the system. The following VTRM DDS fields must be updated:

- Number of displays (set to one)
- Display identifier
- IODN of the display device driver
- IOCN of the device-specific module (if any)
- Number of fonts added (if any)
- IOCN of any added fonts.

Figure 3-41 on page 3-203 shows the format of the VTRM DDS.

VRM install provides an IOCN and IODN for the device driver and places them into the DDS. Any additional code modules are assigned the next largest IOCN and are processed in the order in which they appear on the command line.

An **Attach Device SVC** is then issued for the control program, and the program receives control at its Initialize Device entry point. Any tasks required by the developer can be performed here. A **Detach Device SVC** is then issued, and the control program gets control at its I/O Initiate entry point with a detach queue element in register 3. The Terminate entry point is also called here. The control program must dequeue the detach queue element and return a zero return code (for successful completion) from both the I/O Initiate and Terminate entry points.

After successful completion of these steps, a **Define Device SVC** is issued for the IODN allocated for the display device driver. VRM install then takes the updated copy of the VTRM DDS and redefines it to the system. At this point, VRM install should be able to write to the non-base display.

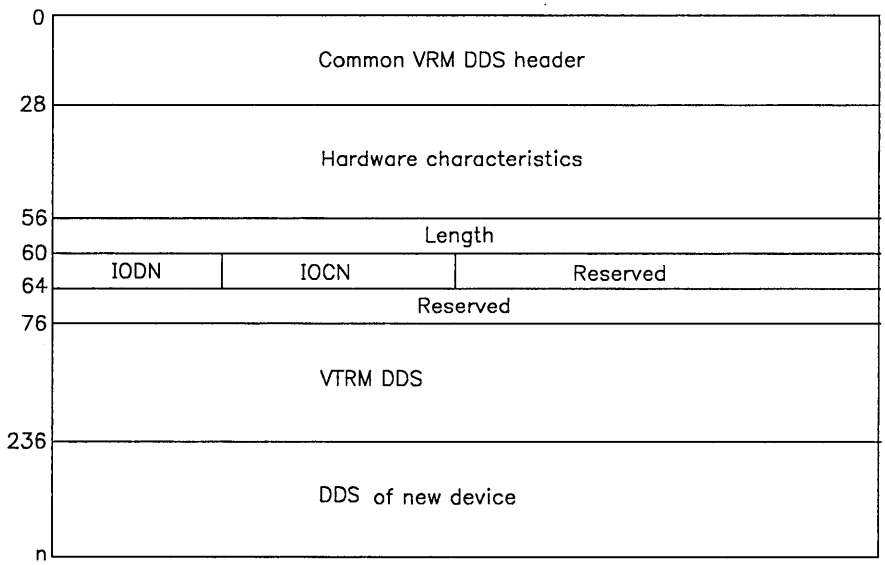


Figure 1-7. Control Program DDS. Note that bytes 56 through n represent the device characteristics section of the control program DDS, even though this section contains two other complete define device structures.

The fields in the control program DDS are defined as follows:

- Common VRM DDS header

  This section includes the IODN and IOCN of the control program, the device type and name, and the applicable offsets to other sections of the DDS. See Figure 2-2 on page 2-5 for a detailed definition of these fields.

- Hardware characteristics

  This section describes the physical hardware features associated with the component, such as the internal device type, pertinent addresses, interrupt definition, and so on. See Figure 2-3 on page 2-6 for more details.

- Length

  This field indicates the length of the control program DDS from offset 56 to n (the control program device characteristics section).

- IODN

  This is the IODN allocated by VRM install for the new display device driver.

- IOCN

  This is the IOCN allocated by VRM install for the new display device driver.

- VTRM DDS

  This field is a copy of the VTRM DDS described in "Virtual Terminal Resource Manager" on page 3-202. Depending on the type of code or device being added to the system, certain fields in the VTRM DDS must be updated.

- DDS of new device

  This field is filled in by the control program and contains the DDS of the device being added to the system. It is a variable length field (indicated by a parameter of the **vtcp** command) because different devices have different requirements for the hardware characteristics and error log fields.

Errors:

Errors that occur with this command cause a two-digit code to flash in the LEDs. This code is associated with a message. These messages are described in *IBM RT PC Problem Determination Guide.*

The following error conditions may occur with **vtcp**:

- An I/O error may result from use of a bad diskette.
- A file referenced on the command line was not found on the diskette.
- The **Define Device SVC** failed.
- The **Attach Device SVC** failed.
- A **Define Code SVC** cannot be issued for a module on the diskette due to incorrect format.
- Insufficient memory to complete the command.

Revising Code in the VRM

Each licensed program product (such as the VRM) shipped by IBM for RT PC contains a ***history file***. This file indicates the version, release, and level of the product code. As software is revised to add function or correct errors, the program product in use on the machine must be updated or reinstalled.

Revisions to existing software can be done either by reinstalling the entire program product or updating specific modules of the product. The following section describes the facilities used by IBM to enable updates to program products installed on the VRM minidisk. Software developers are not obligated to use these facilities in the same way IBM does, or to use them at all. However, non-IBM software developers must be aware of how VRM install uses and checks certain areas of update diskettes before the revisions can be added to the system.

The History File

IBM recommends that software developers include a history file on program products used on RT PC. One advantage to doing this is that, when a program product with a history file is installed on the VRM minidisk, an entry is made in the VRM history file to reflect the product. If the entire VRM must be reinstalled for any reason, the user has a record of the program products that must also be reinstalled.

A history file should contain two records, a c and a t record, and each record must be exactly 80 bytes in length. "Record Types" on page 1-26 defines each record type. Any unused fields in a record should be padded with blanks. The last character in each record must be a newline character and the data cannot contain any tabs.

The history file must be contained on the same diskette as the rest of the program product code. When a program product is installed on the VRM minidisk, the history file from the diskette is copied to the VRM minidisk. This is done automatically, so do not issue an **addf** command. The file must be called **/lpp/lpp_name/lpp.hist**. **lpp_name** is a short name (maximum of 8 characters) for the program product and should match the LPP name field of the c record. Only one history file is allowed per diskette, and restrictions apply to the use of the file name **/lpp**. See "Restrictions on /lpp Files" on page 1-28 for the details on these restrictions.

Note that the path names given for the history and update files are the path names from the root of the file system on the diskette or VRM minidisk.

The fields of the 80-byte history file are defined by IBM as follows:

Bytes	Description
1	Record type This byte represents the type of record on the history file. For history files on the VRM minidisk, this byte can contain a lowercase c, v, t, or asterisk (*). "Record Types" on page 1-26 describes each record type in more detail. The following field descriptions apply only to the c and v record types.

2 Blank

3-10 LPP name

This 8-character field should match the **lpp_name** field in the path name of the history file. Any valid AIX file name can be used for this 8-character field.

11-16 Reserved

17 Blank

18-19 Version

Valid characters are the numerals 0 through 9.

20 Period (.)

21-22 Release

Valid characters are the numerals 0 through 9.

23 Period (.)

24-27 Level

Valid characters are the numerals 0 through 9. Bytes 18 through 27 indicate the version, release, and level of the program product. The IBM convention for the level field, which is initially set to 0000, is to increment it by 10 for each update.

28 Blank

29-30 Day

Valid characters are the numerals 0 through 9.

31-32 Month

Valid characters are the numerals 0 through 9.

33-34 Year

Valid characters are the numerals 0 through 9.

Bytes 29 through 34 indicate the date. For c records appended to a history file or v records added to the VRM history file, the date field is set by the system clock when the program product is updated or installed. The date field in the original c record of a program product is not modified.

35 Blank

36-43 System user's name

Valid characters for this field are the numerals 0 through 9 or the letters a through z (upper- or lowercase).

If the VRM minidisk is changed from the AIX Operating System, this field indicates the log name of the user making the change. When the VRM minidisk is changed by VRM install, this field equals the characters VRMINST.

44 Blank

45-79 Comments

80 Newline character

Record Types

The record type of a record in the history file is indicated by the character in the first byte of the record. Valid characters for this byte are a lowercase c, t, v, or an asterisk (*). Each type is defined as follows:

c indicates a committed record.

The history file for a program product diskette should have as the first record in the file a c record that documents the version, release, and level of the program diskette. Whenever updates are made to the program product, a new c record indicating the level of the updates is appended to the program product's history file on the VRM minidisk. This record has the 80-byte format described previously.

t indicates a title record.

The history file for a program product should have a title record for the second record in the file. The title record is simply a name that can be more descriptive than the 8-character LPP name field in a c record. Bytes 3 through 32 contain this descriptive title. Titles less than 30 characters long should be padded with ASCII blanks. An example of a title record for the VRM might be:

```
t Virtual Resource Manager
```

v indicates a VRM minidisk update.

A v record appears only on the VRM history file (**/lpp/vrm/lpp.hist**) for program products that have made updates to the VRM minidisk. This record type has the same 80-character format as the c record. The date field in a v record is updated by the system clock, however, and the user name field is set to VRMINST. If a v record already exists in the VRM history file for the program product, the version, release, level, and date fields of the v record are updated.

If the VRM is reinstalled for any reason, the user receives a message based on the v records in the VRM history file. This message informs the user of program products that should also be reinstalled.

Note: If a program product without a history file is installed on the VRM minidisk, the message indicating the program products that must be reinstalled with the VRM shows a name of 'UNKNOWN'.

* indicates that this record is a comment.

A comment record must also be 80 bytes in length.

The Update File

History files are useful when an entire program product needs to be revised. However, developers may choose to send out updates of selected modules within a program product. In this case, the diskette contains only the code that is being revised. No history file is necessary. Instead, an ***update file*** on the diskette contains an abbreviated form of the information contained in a history file. Only the first 17 characters of the update file are processed; the 18th character must be a newline character.

Whenever an update is made to a program product on the VRM minidisk, the product's history file on the VRM minidisk is updated with a new c record that indicates the current version, release, and level of the product in use on the system.

The new c record does not replace any existing records, but is appended to the end of the history file. The last c record in the history file contains the latest level of updates that have been installed for a particular program product.

The update file must be called **/lpp/lpp_name/updlevel** where **lpp_name** is the name of the LPP as it appears on the VRM minidisk. The fields of the file named **updlevel** are defined as follows:

Bytes	Description
1-8	Program name
9	Blank
10-11	Version number
12-13	Release number
14-17	Level number.
18	Newline character (0x0A)

Update files have restrictions not associated with history files. For example, an update diskette (**/lpp/lpp_name/updlevel**) will not be processed unless a corresponding history file (**/lpp/lpp_name/lpp.hist**) already exists on the VRM minidisk.

Also, the update diskette will not be processed unless the version and release of the update file are equal to the version and release of the history file. The level field on the update file must then be greater than the level shown on the last c record for the program product history file on the VRM minidisk. For example, if the update file indicates 01010030 for the version, release, and level fields, and the latest c record in the product's history file on the VRM minidisk shows these fields equal to 01010050, the update is unsuccessful. This avoids inadvertent replacement of a current level of program product code with a previous level of code.

Restrictions on /lpp Files

The file name **/lpp** on a program product diskette is a reserved name. If this name is used for a history or update file, the following restrictions apply:

- The file **/lpp** must be a directory file.
- Only one entry in **/lpp** is allowed (with the exception of the dot (.) and dot dot (..) entries for the directory itself and its parent).
- The single entry in the **/lpp** directory file must be 8 characters or less and should be an exact match of the program product name field (**lpp_name**) of the c record. The file **/lpp/lpp_name** must be a directory file.
- Only one entry in **lpp/lpp_name** is allowed. The single file in the **/lpp/lpp_name** directory file must be either a history file (**/lpp/lpp_name/lpp.hist**) or an update file (**/lpp/lpp_name/updlevel**).

Chapter 2. VRM Device Driver Concepts

CONTENTS

About This Chapter

This chapter describes the role of device drivers in the VRM. Topics discussed include VRM device driver command and interrupt conventions, the format of device driver modules, and some general programming considerations for those who wish to add device driver code (or any code) to the VRM. Also discussed are the steps necessary to convert an **a.out** object module to a VRM object module.

Please note that C language examples of the following device driver components are included in the directory **/usr/lib/samples**, which can be installed from the Extended Services diskettes of the AIX Operating System:

- Printer device driver
- Device-specific module for a display device attached to the IBM Color Graphics Adapter
- A loadable power-on self test (LPOST)
- Diskette device driver (provides an example of a device driver that uses direct-memory access).

The task of adding code to the VRM assumes a general understanding of systems programming and should be attempted only by qualified individuals. Whenever code with supervisor state privilege is added to the VRM, it is possible to corrupt the integrity of code and files and have a negative impact on system performance.

If, in the course of adding code to the VRM, you determine that you need to write a corresponding AIX Operating System device driver or want to revise the AIX Operating System files that configure the system at IPL, you should also have *IBM RT PC AIX Operating System Technical Reference.*

Device Driver Interfaces

A VRM device driver provides a queued interface to a physical hardware device. The driver accepts commands from a higher level (either a VRM device manager or a virtual machine operating system device driver) and generates the I/O instructions required to perform the enqueued request. A device driver is also responsible for handling interrupts (usually for I/O completion or exception conditions) from the hardware.

IBM supplies device driver code for most common device types (printers, diskette drives, tape drives) and device management code for complex devices (display terminal, fixed disk). This code is defined at system IPL time. Any additional or supplemental code, such as another display font, a new protocol process, or even a new set of device managers and drivers, must be explicitly defined with a **Define Code SVC** so the code is known to the VRM.

Define Device Structure

The format of device driver code is similar for all types of drivers. "Common Routine Interface" on page 2-14 describes the required and optional entry points for device driver modules. Drivers differ mainly by the characteristics of the devices they control. That difference is manifested in the define device structure (DDS).

Figure 2-1 shows the basic structure of a DDS.

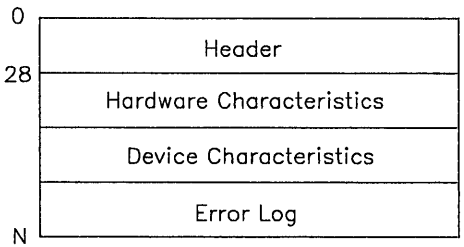


Figure 2-1. Define Device Structure

Notice that only the header is a constant-length field (28 bytes). The length of the hardware characteristics field may differ from one driver to the next depending on the number of interrupts the driver supports and whether the device has any ROM. Other DDS fields, such as the device characteristics and error log fields, vary in length and application for each particular driver. Some drivers do not even use the error log fields, for example.

The DDS is one of the structures used to communicate information between the virtual machine and the VRM. The virtual machine passes the DDS to the VRM as a parameter when it issues a **Define Device SVC**.

DDS Header

The DDS header, which occupies the first 28 bytes of the structure, has the same format for all drivers. Figure 2-2 shows the header structure.

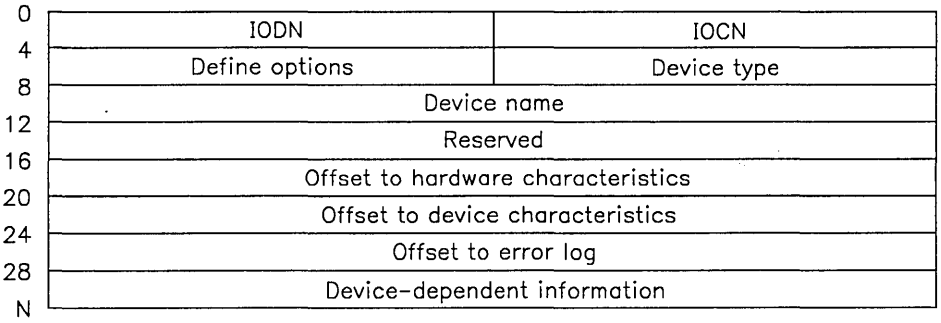


Figure 2-2. DDS Header

See **Define Device SVC** in *VRM Programming Reference* for a detailed discussion of all DDS fields.

A variable-length field of device-specific information follows the header. This field includes the hardware characteristics, device characteristics, and error log.

DDS Hardware Characteristics

Figure 2-3 on page 2-6 illustrates the hardware characteristics structure.

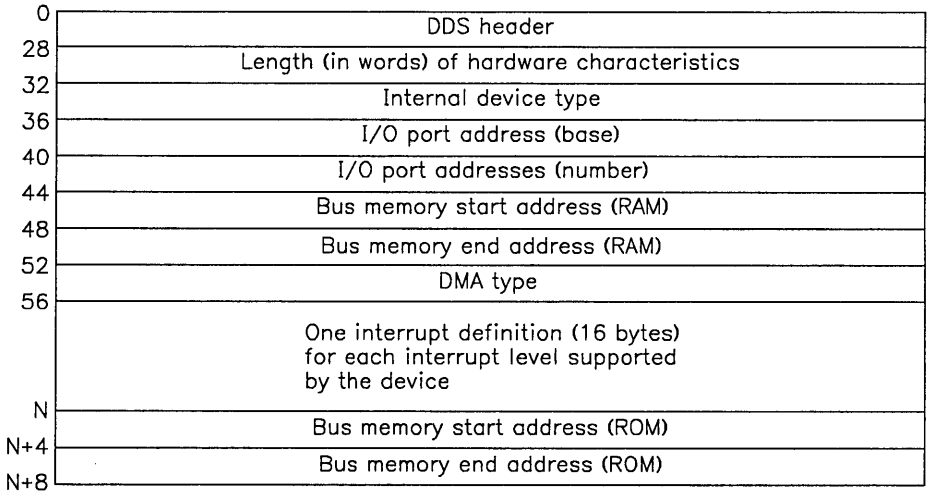


Figure 2-3. DDS Hardware Characteristics

DDS Device Characteristics

Device characteristics fields vary widely for different devices. To see some examples of device characteristics fields from IBM-supplied device drivers, see Chapter 5, "IBM Predefined Device Drivers" on page 5-1.

DDS Error Log

The error log section of a DDS is used to report various types of errors that may occur during an I/O operation. Some of this information is used by the **_errvrm** subroutine to create error entries. These entries are sent to error log files which can be formatted into readable error reports. The error log section can also be used by the **Query Device SVC** to obtain information.

The design of the error log depends on the needs of the device driver that uses it. Some device drivers have relatively simple types of error information, and therefore their error log may consist of only a few words of error data. A relatively simple implementation of a DDS error log is that used by the streaming tape device driver (shown in Figure 5-21 on page 5-118). The error logs of other device drivers may be much more extensive. For example, device drivers that perform error recovery operations, such as the fixed-disk and diskette device drivers, contain much more information. There are two basic formats for this type of error log, a short or a long form. The short and long types are described in the following sections.

Short Error Logs

One type of error log is called a short error log. This is the type of error log used by the diskette and fixed-disk device driver. Figure 2-4 on page 2-7 shows an example of a short error log.

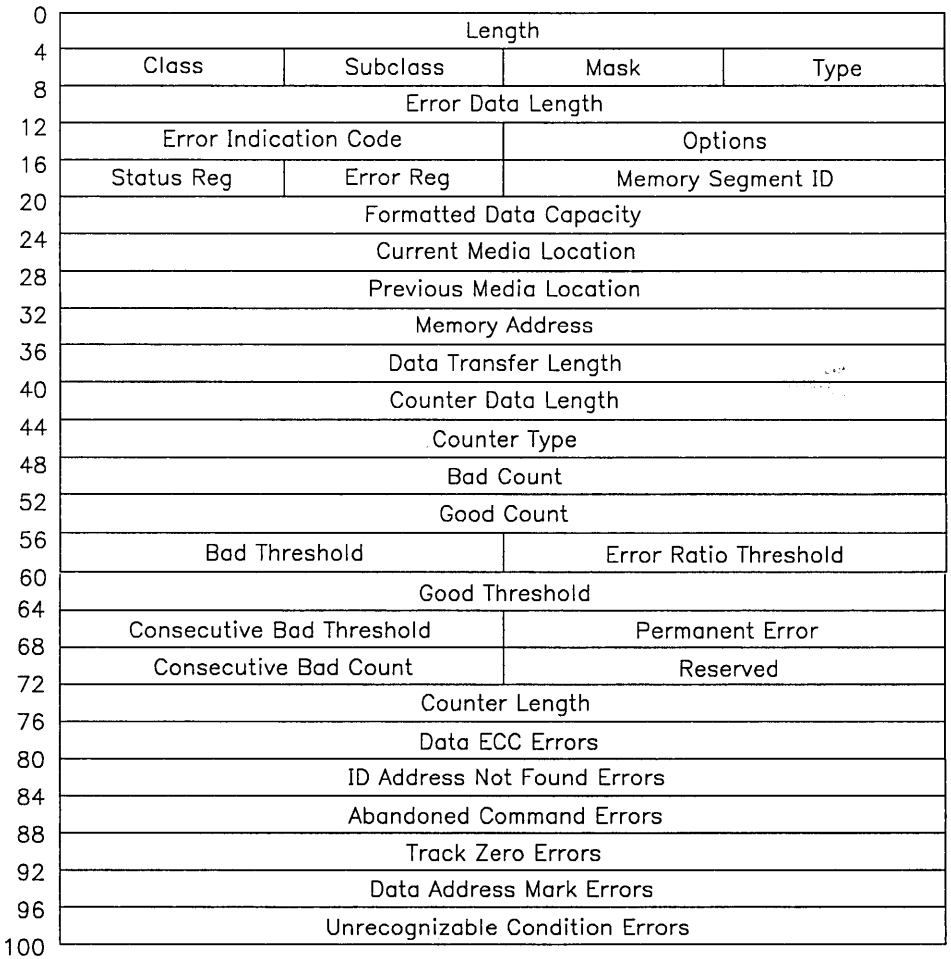


Figure 2-4. Short Error Log Format

A short error log is logically divided into four sections, each of which is used for a different purpose. The first section is always used when an error occurs. It contains three pieces of information:

- The overall length of the error log. This field remains the same for all errors encountered by a particular device driver.

- The error ID, which is composed of the class, subclass, and mask fields. The error ID is part of the data used to create an error entry.
- The type of error entry that is being generated. The value in this field depends on the type of error encountered.

The second section begins with the error data length field and is only used when an error occurs during the current I/O operation. This section is usually six to eight words in length and contains device-specific information or current operation information.

The third section begins with the counter data length field and is only used when an error causes a predefined error ratio threshold to be exceeded. The method used to calculate the error ratio is defined later in this section. For now, just note that this section is used when the device driver falls below a specified level of performance.

The second field in this section is the counter type field. The first bit (bit 0) in this field must be set to 1 to indicate that this is a short error log. The remaining bits in the counter type field are assigned to the various error counters that are maintained by the device driver. When an error occurs that causes the error ratio threshold to be exceeded, the proper bit is set in the counter type field to indicate which error occurred. The remaining fields in this section contain information about the error counter identified in the counter type field. This information includes the number of errors encountered, the number of successful operations performed, the thresholds that determine when to check the error ratio, and the error ratio that is considered acceptable for the device driver.

The fourth section begins with the counter length field and is also only used when an error causes a predefined error ratio threshold to be exceeded. This section contains a list of error counters that identify the total number of errors encountered for each type of error that can occur for the device driver. The error counter value for each type of error is one word in length.

Long Error Logs

The other type of error log is called a long error log. This error log contains the same first three sections as a short error log, but it does not contain the fourth section. Instead, it repeats the third section for each type of error counter maintained by the device driver. Note that each repeated section does not include the counter length or counter type fields.

The counter type field is used to identify a long error log. The first bit (bit 0) in the counter type field must be set to 0 to indicate that this is a long error log. The remaining bits in the counter type field are assigned to the various error counters that are maintained by the device driver. When an error occurs that causes the error ratio threshold to be exceeded, the proper bit is set in the counter type field to indicate which error occurred. The bit position is used as an index to locate the rest of the error data associated with that error counter.

Using Error Counters in Device Drivers

Due to the nature of hardware devices, some I/O operations are unsuccessful the first time they are scheduled by the device driver. This is not necessarily an error condition. It may be possible to perform a retry on the operation and have a successful completion; however, the device driver should keep track of errors that cause retries and generate an error entry if the number of errors becomes excessive. This section describes one method that can be used to track errors and report data to the error log.

There are two basic types of errors:

- Current operation errors

  These errors appear for the current I/O operation. You can define a set of variables that state when an error entry should be generated. For example, you can say that any operation that requires 0 to 2 retries completed without incident, 3 to 9 retries is considered a temporary error, and 10 retries indicates a permanent error. These types of errors are reported in the second section of the error log.

- Cumulative errors

  These errors accumulate over time. The device driver keeps track of all of the errors that have occurred for each I/O operation, and when certain conditions occur, computes the ratio of errors to successful operations. If the error ratio is too high, the error counters and associated data are reported to the error log and an error entry is generated. These types of errors are reported in the third and fourth sections of the error log.

A device driver can use the following counters and variables to determine when to report an error and which type of error to report. These counters and variables are kept in the device driver's working area and should be available at all times during an I/O operation. The values are written to the error log when an error ratio threshold is exceeded:

Consecutive bad count
: This is a counter that is reset to 0 at the beginning of each I/O operation. It is incremented each time a retry is performed for any reason.

Error counters
: One cumulative error counter for each error type. Each time a retry is performed, the device driver determines the reason for the retry and increments the appropriate error counter. A counter is reset to 0 if it causes the error ratio threshold to be exceeded.

Good count
: One good count counter for the total number of successful operations that have occurred. This value is used to compute the error ratio threshold.

Error ratio threshold
: The value which will cause the error counter data to be reported. This value is obtained by dividing an error counter by the good count. A typical value for this is 5, meaning 5 percent. The error counter used to compute the error ratio is the one that reaches the bad threshold.

Consecutive bad threshold
: The number of consecutive retries that must occur before an error of any type is reported.

Permanent error threshold
: The number of errors that must occur before a permanent error is reported.

Bad threshold
: The number of retries that must occur before checking the error ratio threshold. In general, this value should be the same as that used as the permanent error threshold.

Good threshold
: The maximum number of good operations which can be used to compute the error ratio. If the good count counter reaches this value, it is reset to 0 by the device driver. This ensures that the good count value does not become so large that the error ratio becomes statistically invalid. If the good count was always incremented, it could eventually reach a level where the error ratio would never be large enough to exceed the error ratio threshold.

The following scenario describes how the error counters are used to monitor I/O operations:

1. When the system is initialized, all of the error counters are set to 0.
2. The device driver receives an I/O command, resets a current operation retry counter to 0 and attempts to perform the command.
3. If the device driver encounters an error that cannot be retried, a permanent error is reported and the operation is abandoned.
4. If a retry is required, the current operation retry counter is incremented by 1 and the error counter corresponding to the type of error that caused the retry is incremented by 1.

   The following sequence is performed before each retry:

   a. If the number of retries is greater than or equal to the bad threshold value, and the good count counter is greater than or equal to the minimum value needed to obtain a statistically meaningful error ratio, an error ratio is computed by dividing the bad count by the good count. The bad count is the value of the error counter that caused the retry.
   b. If the error ratio is less than the error ratio threshold, the error counter data is not sent to the error log.
   c. If the error ratio is greater than or equal to the error ratio threshold, the error counter data is sent to the error log. The error counter that caused the ratio to be exceeded is then reset to 0.
5. If necessary, the device driver attempts to perform the operation again. These steps are repeated until the operation completes successfully or a permanent error condition is encountered.

6. At some point, the operation will be completed, either successfully or unsuccessfully. Operation results include:

   - Successful completion, no error reported
   - Successful completion, temporary error reported

     This means that the operation required some number of retries, but managed to perform its task successfully.
   - Unsuccessful completion, permanent error reported

     This means that the operation required so many retries that it exceeded its limit, or the operation encountered an error that did not allow retries.

7. If the operation finishes successfully, regardless of the number of retries, the good count counter is incremented by one. If this value reaches the good threshold, the good count is reset to 0.
8. Finally, the device driver signals completion of the operation.

Input/Output Operations

Two SVC instructions control hardware input/output operations. They are:

- **Start I/O SVC**
- **Send Command SVC**

Whether a device uses a **Start I/O SVC** or a **Send Command SVC** depends to a large degree on whether the designer of the code wants the command control block (CCB) facilities provided with **Start I/O SVC** or the efficiency of the **Send Command SVC**. Both SVCs send similar types of data, but because **Send Command SVC** passes its parameters in registers, it is more efficient than the **Start I/O SVC**. The **Start I/O SVC** allows multiple data buffers to be associated with a single I/O operation because CCBs can be linked in a chain. Figure 2-5 on page 2-12 shows the format of the CCB and the link bit (L). A link bit that equals one means multiple command elements are being sent.

One use of this capability is to perform ***scatter*** and ***gather*** operations. Scatter means reading data from a device and 'scattering' it to different memory locations. Gather means taking data from different memory locations and 'gathering' it to write to a device.

CCBs can also be used by a virtual machine to send a list of buffers to a device driver for use in subsequent operations.

For most operations, however, the **Send Command SVC** is sufficient and should be used due to its lower overhead.

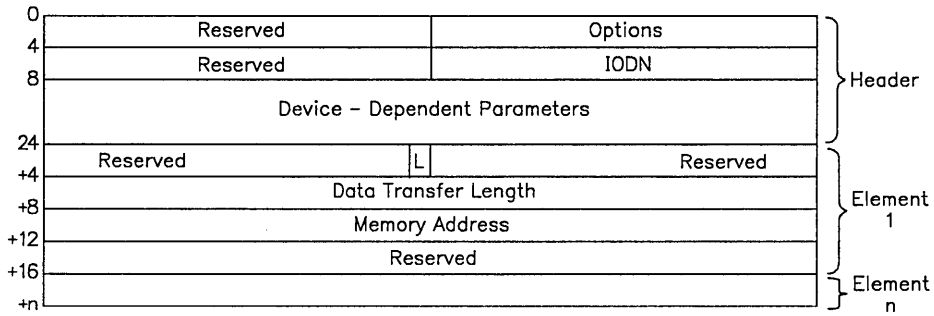


Figure 2-5. Command Control Block

See **Start I/O SVC** in *VRM Programming Reference* for more information on CCBs.

Operation Completion and Status Values

When an I/O request completes or cannot be performed due to some error condition, the device driver notifies the operating system. This information is returned to the operating system in the form of a program status block (PSB). The PSB for the **Start I/O SVC** is shown in the following figure:

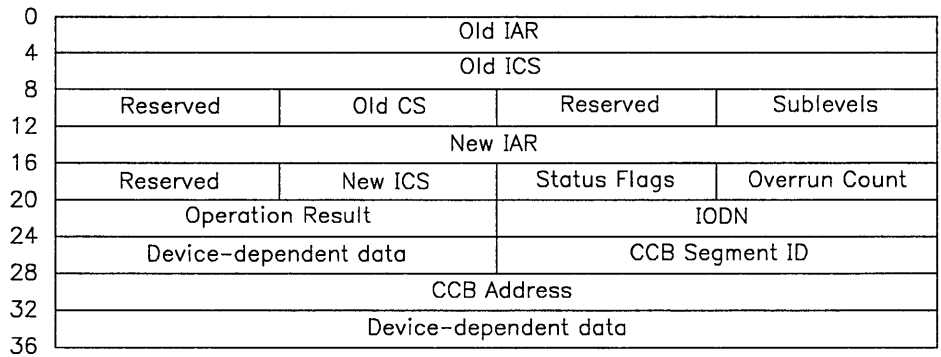


Figure 2-6. Start I/O Program Status Block

I/O requests initiated with the **Send Command SVC** that generate interrupts also signal the operating system of I/O completion or detected errors with PSBs. The format of the **Send Command SVC** PSB, shown in the following figure, is somewhat different than the PSB of the **Start I/O SVC**.

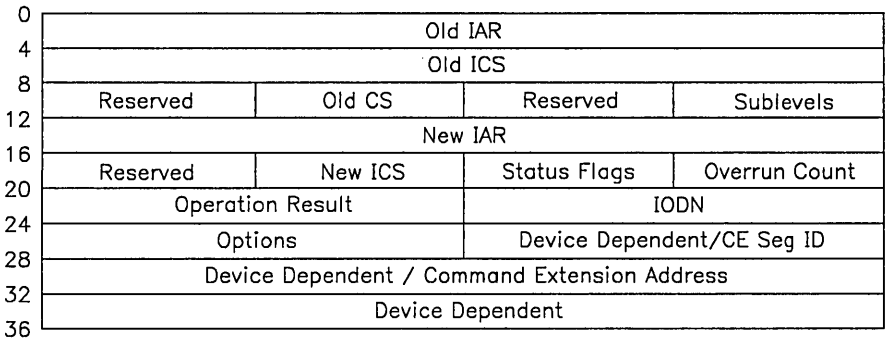


Figure 2-7. Send Command Program Status Block

See **Send Command SVC** in *VRM Programming Reference* for a detailed definition of PSB fields.

Regardless of the PSB format, the PSB fields that are pertinent to the virtual machine on return are the status flags and the operations results.

These fields may be defined differently for different device types and implementations. To see how IBM defines these fields for supported devices, see Chapter 5, "IBM Predefined Device Drivers" on page 5-1.

Common Routine Interface

The VRM presents a standardized call to all modules, regardless of whether the module is part of the base VRM or is subsequently installed. Both device manager processes and device drivers must conform to this common interface, although the two module types require some different entry points.

VRM modules consist of several sections (subroutines), each with a specific function to perform. Certain functions are required of VRM modules and certain other functions may be optionally defined. Some of the functions that can (or must) be performed by device driver or manager modules include definition and initialization, I/O initiation, parameter checking, and exception handling.

Each module has a default main entry point from which all its defined functions can be accessed. In addition, you can use the **_change** routine to define additional entry points to specific functions.

The following table lists entry points that can (or must) be defined for VRM device manager or device driver modules. Optional functions are followed by an asterisk (*).

Device manager modules

- Process initialization
- Queue parameter checking *
- Exception handling (including timer notification) *

Device driver modules

- Device definition (default main entry point)
- Device initialization
- Device termination
- I/O initiation
- Interrupt handling (second level)
- Queue parameter checking *
- Off-level input/output processing *
- Exception handling (including timer notification) *

For both device driver or manager modules, the parameter conventions of the call from VRM are the same. Input parameters of a call from the VRM must include a type code and the address of some data area. The type code indicates the function being called and the data area contains the address of operation-specific data. The data area varies with the specified type code and may be a queue element, define device structure, exception mask, interrupt level, and so on.

The calling register conventions for a call to a device driver or manager are defined as follows:

GPR2 = Type code

GPR3 = Address of the data area

GPR4 = Length of the data area.

GPR2 contains the return code from a call. The normal return code from a call is 0. Codes other than 0 indicate further action by the VRM. The define device routine for device drivers returns -1 if the supplied DDS is invalid.

For C language modules, the common routine interface is specified as follows:

```
int main (op_type, op_data, length)
unsigned int op_type;
char *op_data;
int length;
```

For synchronous queued operations, the 16-bit operation results field from the acknowledgement queue element is sign-extended to 32 bits. This 32-bit value is used as a return code. Within the VRM, the value returns to the caller of the enqueue function. For virtual machines, the value returns in GPR2 after an SVC instruction.

The following sections provide details of the common routine interface and pertinent information for device drivers and device managers, respectively.

Interface for Device Drivers

The VRM tries to perform as many common device driver functions as possible. It handles such system details as generating virtual interrupts, allocating storage for code and data areas, managing the interrupt-controlling hardware, and overseeing interactions between components. Ideally, the VRM allows device driver subroutines to be small, simple, and fast. The VRM does not handle device-specific functions, however. You must handle unique device requirements within a device's driver module.

The following chart summarizes the common routine interface for device drivers. Subsequent sections provide details you may find helpful when coding device drivers.

Note that the values used in the following charts represent decimal values.

Function	Type	Data Area	Ret. Code
Check parameters	21	Send command queue element	0 or ≥256
Check parameters	22	Start I/O CCB queue element	0 or ≥256
Check parameters	23	General purpose queue element	0 or ≥256
I/O initiate	30	Queue element	0
Interrupt handler	40	Bus interrupt level	0, 2 or pointer to DDS for off-level request
Exception handler/ timer notification	50	Exception mask	0 or -1
Define device	60	Define device structure	DDS address or -1
Initialize device	61	VRM passes the structure shown in Figure 2-9 on page 2-17.	0
Terminate device	62	6-byte field made up of DID and IODN	0
Off-level processing	70	DDS address	0

Figure 2-8. Device Driver Module Entry Points

Initialize Device Area Data

The VRM passes the address of the following data structure to the initialize device entry point.

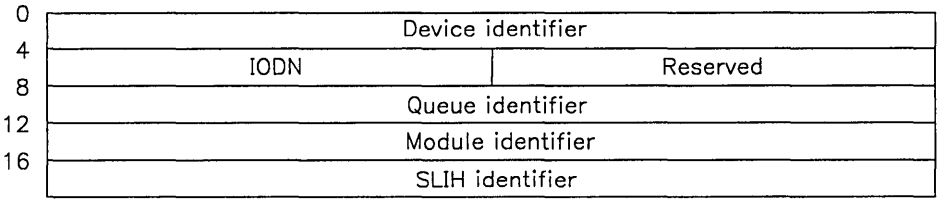


Figure 2-9. Initialize Device Data Structure

Entry Point Types

All calls to device driver modules go to the module's main entry point unless the **_change** routine is used to define an alternate entry point. When a call arrives at a module, the type parameter is examined to determine the requested function. The **_change** routine can also be used to define additional entry points. If you specify additional entry points, the module can be entered directly at the requested function without having to branch from the main entry point. A brief discussion of each entry point type follows.

Check Parameters

This subroutine can check parameters for each command queue element before the element is placed into a device's queue.

The advantage of using check parameters subroutines is that you can filter out unacceptable commands before they are enqueued to a device. The subroutines look at the queue elements and return 0 if the element is acceptable or a value ≥ 256 if it is unacceptable.

The VRM will not place the queue element in a device's queue if it sees a non-zero return code. Check parameters subroutines have addressability to the queue element, but not to buffers or the CCB. You can use **_lsr** to load segment registers 1 or 2 to see these areas.

I/O Initiate

A virtual machine uses either the **Start I/O SVC** or the **Send Command SVC** to initiate I/O operations. A device manager can utilize the VRM routine **_enque** for the same function.

The VRM maintains a commands queue for each driver. This queue is set up when the first user attaches to the device with the **Attach Device SVC**. The I/O initiate subroutine is called once for each element in the commands queue. See *VRM Programming Reference* for a description of the command queue elements.

The main task of this subroutine is to send the queued commands to the adapter. If the queue is empty, the VRM calls the I/O initiate function immediately. If the queue is not empty, the VRM waits until the device driver completes the current command (by calling **_deque**) before calling the I/O initiate function.

Usually, a driver processes each command to completion, then takes the next command until the driver's commands queue is empty. A copy of the top queue element is passed to this subroutine as a parameter when I/O initiate is called. Because this copy exists on the stack, it will disappear when I/O initiate returns to the VRM. You may want to have access to this data in the interrupt subroutine. You can do this in one of two ways. They are:

- Copy the 32-byte queue element to a static data area.

  This may be the simpler of the two methods. However, a driver may have trouble managing the static data areas if it tries to keep track of multiple commands at once.

- Use the **_readq** function.

  This runtime routine returns a copy of the top element of a specified queue. *VRM Programming Reference* describes this routine.

One other function performed by the I/O initiate routine is to call **_deque** when the received queue element is a type 4 (a control queue element).

Interrupt Handler

The main processor receives interrupts on several distinct lines known as levels. Each interrupt level has an associated ***first-level interrupt handler*** (FLIH). Within each level, there are four device classes that can return interrupts. The device classes are defined as follows:

0 = Devices that overrun but have no recovery mechanism.
1 = Devices that overrun but can recover.
2 = Performance-sensitive devices that do not overrun.
3 = Devices that do not overrun and are not performance-sensitive.

When an adapter sends an interrupt to the processor, the FLIH corresponding to that device's interrupt level is called. Because multiple devices can share the same interrupt

level, each device also has a ***second-level interrupt handler*** (SLIH). Figure 2-10 shows the relationship between FLIHs and the SLIHs they poll.

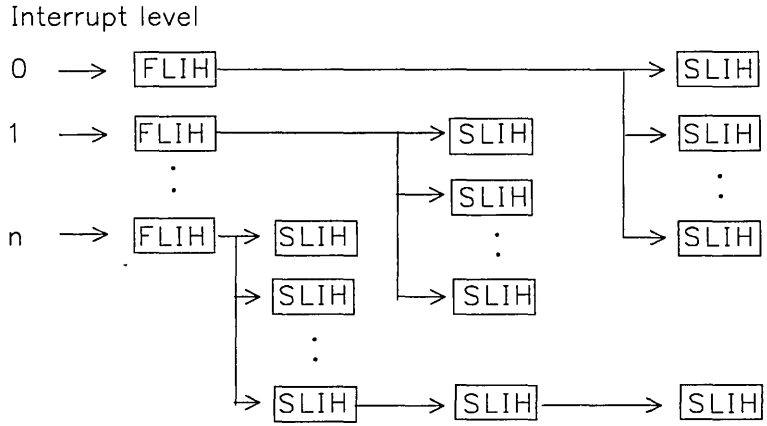


Figure 2-10. VRM Interrupt Handling

After the FLIH saves the state of the process (or other interrupt level) that was interrupted, it polls all the SLIHs associated with that particular interrupt level until the proper SLIH handles the interrupt.

When polled, the SLIH checks its device's status to determine if its device generated the interrupt. If so, the SLIH processes the interrupt and sends a zero return code to the FLIH. If its device did not cause the interrupt, the SLIH returns a 2 to tell the FLIH to continue polling the other SLIHs on that interrupt level.

To handle an interrupt, the SLIH may just reset the device and send it the next command. If the work requested by the current queue element is complete, the SLIH dequeues the element with **_deque**. Two parameters to **_deque** are significant here, the QID and the 32-byte structure used to generate the acknowledge queue element. Acknowledgements are sent to the component that issued the command. A virtual machine receives acknowledgement by way of a virtual interrupt. A device manager receives an acknowledgement queue element. Either way, the VRM generates and routes the acknowledgement. The device driver must simply supply the status data to **_deque**.

This status data includes a halfword operation results field. Zero indicates successful completion. Unsuccessful completion means the left-most bit of this halfword is turned on and an error code in the range 1 to 32,767 is returned. This value is sign-extended when it reaches the virtual manager or device manager that issued the command, thus becoming a negative number in the range -1 to -32,767. The system reserves values 0x8000 and 0xFFFF.

You can suppress the acknowledgement from **_deque**, although this is not usually necessary. However, consider the example of a communications device driver. A virtual machine or device manager may send empty receive buffers to the device driver, using a

command queue element for each. The device driver wants to dequeue these elements to remove them from its queue and process other commands. However, the commands that sent the buffers are not yet complete and will not be complete until the buffers are filled.

The device driver can dequeue the commands with the suppress option. As each buffer fills, the driver calls **_enque** to send the acknowledgement queue element. These are known as solicited acknowledgements since they correspond to a command queue element. A bit in the flag area of the queue element reflects this situation.

A device driver may also generate unsolicited acknowledgement queue elements. For example, a driver for a communications line may detect something on the line and want to notify someone about it. Because no element is in its queue, the acknowledgement of this event must be unsolicited.

Another example of unsolicited acknowledgements is seen through the keyboard device driver. Each time you press a key, this device driver sends an unsolicited acknowledgement.

Note that if the suppress option is used, the device driver is responsible for unpinning the CCB and any buffers associated with a completed command. Ordinarily, the **_deque** routine does this unpinning when generating an acknowledgement. The **_upnccb** routine can be used to unpin a CCB and its buffers. The **_unpin** routine is used to unpin specific pages of memory.

Exception Handlers and Timer Notification

Both of these entry points are indicated with a type code of 50. These functions are called as a result of a device manager using the **_signal** function or as the result of a timer expiration. Timer expiration is the most common use of this function.

Device drivers can use two types of timers, device timers and interval timers. A device timer, with a granularity of 500 milliseconds, detects device timeout conditions. You set this timer the first time you call **_setdvt**. Each time the I/O initiate subroutine gets a new command queue element, this timer starts. The **_deque** function stops the timer, and the **_ctldvt** function can reset the timer to its original value. See *VRM Programming Reference* for a detailed description of these routines.

Use the interval timer when you need a finer granularity of time. This timer can be set in increments of 975.562 microseconds (about 1 millisecond). One drawback with using the interval timer is that you have to call **_settmr** to set the interval whenever you want to use this timer.

The device timer can handle most device driver timing needs, but if you need a finer granularity of time and don't mind the extra overhead, use the interval timer.

Define Device

Creation of a device involves the define device subroutine. The entry point to this function is the module's main entry point. Unlike other entry points, this cannot be changed. This subroutine is called once with a type code of 60 when the **Define Device SVC** is issued. Data passed to this routine includes the define device structure (DDS). The define subroutine must copy the DDS to a static data area in the module, passing the address of this area back to the VRM. When you copy the DDS, you can change certain DDS device type values including:

- The limit on the number of paths
  - Shared devices default to 0 (unlimited number of paths)
  - Non-shared devices default to 1.
- The number of priority levels for the device's queue

  Ordinarily, this value defaults to 1. When you copy the DDS, however, you can specify a value in the range 2-16 for priority levels.
- The internal device flag

  You can set or reset this value when copying the DDS.

You can also change the following values (from their default of zero) in the DDS device option field:

- The maximum concurrent requests per path
- The maximum number of concurrent timer requests
- The limit on the maximum number of concurrent unsolicited interrupts per path.

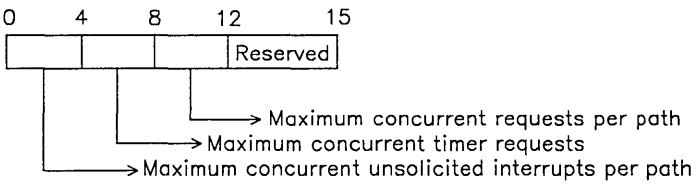


Figure 2-11. Define Options Halfword

Each individual driver defines the parameters that must be contained in the device characteristics section of the DDS. If applicable, the device driver must also update the device characteristics and error handling sections of the DDS. The hardware characteristics section must remain unchanged.

Initialize Device

This function is called as a result of an **Attach Device SVC** and initializes the adapter from fields provided in the DDS. When the initialize device function is called (type = 61), the module is entered at the define device function entry point. This entry point cannot be changed.

This function performs all device-specific initialization. The VRM allocates the device's physical resources, creates its control blocks and queues, calls the define device subroutine, and enables interrupt levels and any required DMA channels.

Because a device is not initialized until attached, system resources are saved and hardware resources can be used more efficiently. With this method, for example, two devices that do not support interrupt sharing can use the same interrupt level (provided the devices are not attached concurrently). Another important reason for not initializing a device until you attach to it is to support the coprocessor. If you allocate serially-reusable devices to a virtual machine, those devices cannot be used concurrently by the coprocessor.

If initialization involves sending a command to the adapter, the initialization function must return control to the VRM, which calls the interrupt handler function when the command completes. If initialization is unsuccessful, you won't find out until the interrupt handler subroutine is called. You can set up an interrupt handler subroutine to send an unsolicited interrupt to the virtual machine to notify it of unsuccessful initialization. However, all you really have to do is wait until the virtual machine sends the first command to the driver. The interrupt handler will then report the status.

Because this function returns only 0 to the VRM, no status information can be sent to a virtual machine by way of other return codes.

Terminate Device

This call is made in response to a **Detach Device SVC**. Calls to terminate a device (type = 62) also enter the module at the define device entry point, and this cannot be changed. This call is made automatically when the last (or only) user of a device detaches from it. The function must do whatever is required to shut down the device.

In order to ensure that a device driver's termination function is not called while an I/O operation is still in progress, the VRM enqueues a control queue element (type 4) after all I/O requests in the queue. The driver knows by convention that a control queue element signifies completion of all pending I/O operations. After the driver dequeues the detach element, the VRM calls the driver's termination routine.

Off-level Processing

This function allows an interrupt handler to return to the VRM quickly and process any lengthy input/output operations off of the interrupt level. Off-level processing makes a big difference when device drivers must handle high-speed asynchronous lines, for example. These drivers must exit from the interrupt very quickly so they do not miss the next interrupt. You must decide, when designing drivers, what must be done with interrupt handler function and what can be done off-level. The VRM keeps track of off-level processing in the device's DDS. The interrupt handling function then returns the address of the DDS scheduled for off-level processing (instead of the usual 0 or 2 return code). A return code that points to a DDS implies the same as a zero return code in that the SLIH acknowledges responsibility for the interrupt. You may want to use the **_sio** (schedule off-level processing) routine if the interrupt handler has several devices that can have interrupts pending on a single adapter.

If you decide to handle some I/O processing off of the interrupt level, the VRM will place the affected DDS addresses in a queue. When no more interrupts are pending, the VRM handles the off-level input/output requests in the queue. If no more interrupts are pending and all off-level I/O requests are handled, the VRM dispatches another process to run.

In the mean time, if an interrupt occurs that reschedules a DDS before it has a chance to run off-level, a "run again" flag is set so the off-level subroutine will be called twice in a row.

Figure 2-12 on page 2-24 shows the entry points into a typical VRM device driver and some of the SVCs that may be received from an AIX Operating System device driver.

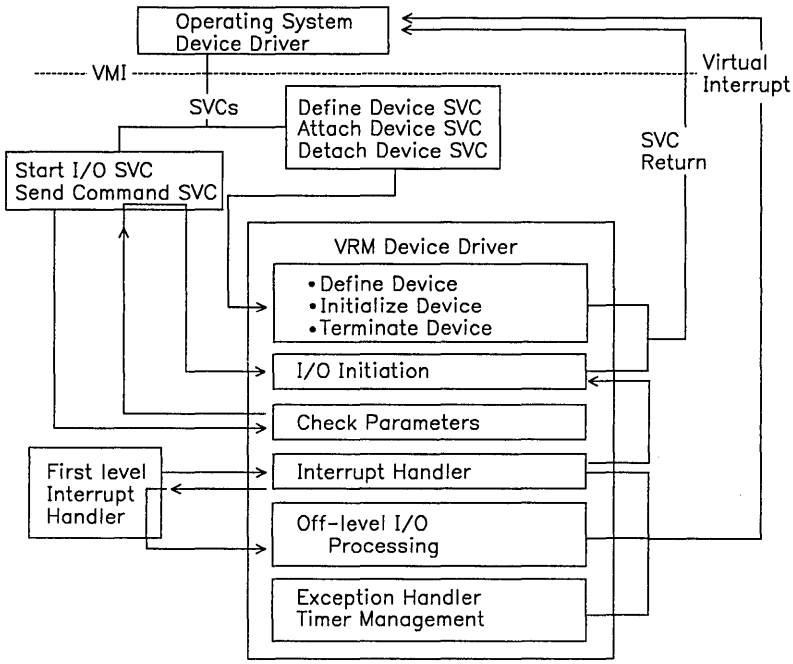


Figure 2-12. VRM Device Driver Functions

Interface for Device Managers

A device manager can coordinate the activities of multiple device drivers and, in some cases, multiple VRM processes.

Most device subsystems can use the device driver model described in the preceding section. More sophisticated subsystems, such as those that involve virtualized devices, may require a device manager. For example, the virtual terminal subsystem merges several devices (display, keyboard, speaker, and perhaps a locator, tablet or light pen). All of these devices, and their accompanying drivers, are used together as a terminal. In addition, this subsystem simulates the existence of multiple terminals that exist on a single set of physical devices.

The following chart summarizes the common routine interface for device managers.

Function	Type	Area Data	Ret.Code
Initialize process	10	Initialization data	Always 0
Check parameters	20	Acknowledgment queue element	≥256
Check parameters	21	Send command queue element	≥256
Check parameters	22	Start I/O CCB queue element	≥256
Check parameters	23	General purpose queue element	≥256
Exception handler/ Timer notification	50	Exception mask	0 or -1

Figure 2-13. Device Manager Module Entry Points

Initializing A Device Manager

The main entry point of the device manager's module is called when a device manager process is initialized. The type code parameter is a 10, and the area data is the manager's DDS. At this time, all initialization activities are performed and the device manager waits for a command.

Waiting for a command involves calling **_wait** or **_waitq**. Both of these calls require an ECB mask or QID as input. The VRM automatically creates a queue for the device

manager when the process is initialized. To determine the ECB mask and QID, use the **_queryi** function.

Processing a Command

When the manager receives a command and is released from **_waitq**, it receives a copy of the command queue element as a parameter. If **_wait** was used, you have to use **_readq** to get a copy of the element.

To remove a command from the manager's queue, use the **_deque** function. Here **_deque** works the same as it does for device drivers, automatically sending an acknowledgement queue element to the sender of the command (unless the suppress option was used with **_deque**).

The device manager can use **_enque** to generate acknowledgement queue elements or to send commands to another device manager or driver.

If a device manager acts merely as a command router (receiving commands and sending them to other components), you should use two queues. Use one queue to receive commands and another queue for receiving acknowledgements after you pass the commands on. The queue that is set up automatically for the process should be used for the incoming commands. Use the **_creatq** function to set up the queue for the acknowledgements. The **_attchq** function then attaches the device manager to the queues of other components. When you specify the acknowledge QID parameter to **_attchq**, use the ID of the second queue.

With two queues, the device manager should use **_wait** to wait for both queues at once. When a command comes in and the manager is released from the **_wait**, the ECB mask can be examined to check for elements in either queue. If so, use **_readq** to look at the top element in either queue.

Exception Handling and Timers

Device managers can use **_change** to set up an exception handler subroutine, just as device drivers can. They can also use the interval timer.

Developing and Adding Code to the VRM

This section describes how and why to add code to the VRM.

Coding a module and installing it into the VRM requires a relatively thorough understanding of the VRM environment. You need to understand the common routine interface definition and the general coding considerations of the VRM as you develop device driver or manager code. If you are familiar with the overview and general coding consideration sections of this chapter, refer to "Installing Code from a Virtual Machine" on page 2-29 for a condensed version of the steps necessary to install code.

The primary reason for adding code to the VRM is to manage either physical or virtual devices. A device driver controls a physical device through its adapter. A device manager process controls multiple device drivers or access to a virtual device.

The VRM supports a set of devices that comprise the minimum system configuration. These devices include a fixed disk, diskette drive, display, keyboard and streaming tape. These devices are set up at VRM IPL time.

The VRM also contains code to support many other devices. These optional devices extend system function. For example, printers, modems, and certain nonstandard displays can be optionally configured to enhance the system. IBM provides device driver code to support some of these devices. You can add these devices to the system with the **devices** command described in *IBM RT PC Installing and Customizing the AIX Operating System.* Chapter 5, "IBM Predefined Device Drivers" on page 5-1 describes the IBM-supplied components, including several device drivers and a device manager.

If the device you want to add to the system is not supported by IBM-supplied code, you can develop and install your own code into the VRM. You must issue a **Define Code SVC** to add code to the VRM. Note, however, that this code does not exist in the VRM after you shut down the machine or re-IPL the VRM. If you want the system to include your new code automatically at each IPL, you must modify the AIX Operating System configuration files **/etc/master** and **/etc/system**. The parameters in these files cause the AIX Operating System to dynamically issue all the SVCs needed to configure VRM device drivers, managers, and protocol procedures at each IPL. For more information on these files, see *AIX Operating System Technical Reference.*

Code that has been added to the VRM by way of the **Define Code SVC** can be used in the VRM environment with the various runtime routines.

For example, a device manager process in the VRM uses runtime routines to create the additional process necessary to handle each instance of a virtual device. A VRM process can create other processes, queues, semaphores, and so on with these calls.

General Coding Considerations

In order to successfully install your own module into the VRM, you must be aware of certain restrictions imposed by the VRM environment. First, the module you install must be capable of communicating with other VRM code. This communication relies on well-defined entry points among all VRM modules. "Common Routine Interface" on page 2-14 describes how to set up entry points for VRM device drivers or managers. Other considerations that are important when developing and installing a module include:

- Modules must conform to C programming language conventions. These conventions include the C language calling sequence and **a.out** module format.
- If you want your new module loaded at IPL time (module permission bits = 440 octal), you must change the permission bits of the **/vrm/ldlist/vrmbase/vrmglu.0011.00** file to 440. The **vrmglu.0011.00** file resolves linkage convention differences for **a.out** modules in the VRM. See "Permission Bit Conventions" on page 1-8 for more information on permission bits.
- The module must be designed to be memory location-independent. Hard coding of addresses is not allowed.
- All programs must execute with address translation on. Neither the code, static data, nor the heap or stack areas are in virtual = real memory. Program execution without address translation yields unpredictable results. However, address translation is on by default.
- Processor I/O interrupts must not be disabled when the path length is long. Device overruns may result. Also, processor I/O interrupts must be enabled for all calls to VRM services (runtime routines). If interrupts must be disabled, it should be for the shortest time possible.
- Processes have a maximum stack size of 60K bytes. This 60K byte maximum includes any stack requirements of the routines called by the process.
- Device drivers have a maximum stack size of 2K bytes. This 2K maximum includes any stack requirements of the routines called by the device driver.
- Exception handler subroutines also have a maximum stack size of 2K bytes. This 2K maximum includes any stack requirements of the routines called by the exception handler.
- C language modules may have only one external reference to an externally-defined symbol. This includes VRM runtime routines, global variables, and any other externally-defined routines or variables that are imported when binding to another C language module using **_bind**. Multiple references are made when there is more than one function in a C language module that refers to the same externally-defined symbol.

Installing Code from a Virtual Machine

This section describes how to prepare a module for addition to the VRM. Note that code you install into the VRM remains part of the VRM only until you power off the machine. In this case, the code is considered 'transient' because, in order to use the same code at another time (another IPL), you must install it again. If you expect to use the added VRM code repeatedly and not just for one IPL, you should revise the AIX Operating System files **/etc/master** and **/etc/system**. The AIX Operating System reads these files at IPL time and automatically issues the SVCs necessary to add a VRM manager or driver to the regular configuration sequence.

Your new VRM module requires a corresponding support driver in the AIX Operating System. If no such AIX Operating System driver exists, you have to create and install one. In this case, revision of the configuration files alone is not adequate to add the AIX Operating System driver to the system. You must also perform another system generation to relink the kernel. For more information on the AIX Operating System configuration files and writing AIX Operating System device drivers, see *IBM RT PC AIX Operating System Technical Reference.*

Regardless of whether you want to install a VRM device driver, device manager, or protocol procedure, the techniques of coding and installing the module into the VRM are the same.

To install a driver into the VRM, you must install the actual code module and also a define device structure (DDS). The DDS contains the definition of the device and the module contains the program (code) associated with the component. The **Define Code SVC** is used to place the module into VRM address space. The **Define Device SVC** passes the address of the DDS to the VRM as a parameter. Note that this added code is placed on the operating system minidisk, not on the VRM minidisk. If you need to add code (such as a new device driver, power-on self test, and so on) to the VRM minidisk, use the **addf** ("addf — Add a file to the VRM minidisk" on page 1-16) or the **mvmd** command described in *IBM RT PC AIX Operating System Commands Reference.* Remember that the permission bits of the `vrmglu.0011.00` file must match the permission bits of the new device driver module.

The following steps describe how to add a C language device driver or manager module into the VRM. Please note that, although the VRM requires that added code be in **a.out** object module format, the VRM itself does **not** use the **a.out** format. The VRM-executable object module format is described in *VRM Programming Reference.*

In order for **a.out** modules to execute in the VRM, they must be converted to the VRM-executable object module format. The conversion procedure is described in the following section.

The following description assumes you have the AIX Operating System and that you want your code configured at each operating system IPL.

Adding Code to the VRM

1. Design and write your module.
2. Compile the module with the AIX Operating System **vcc** command.
3. Convert the output from **vcc** with the AIX Operating System **vrmfmt** command.
4. Revise the AIX Operating System configuration files.
5. Issue the AIX Operating System **vrmconfig** command.

The steps described above are expanded with more detail in the following section.

Please note that you may encounter error or informational messages as a result of compiling and converting a C language module. These messages are described in *IBM RT PC Messages Reference.*

1. Design and write your module.

   First, you design and develop the driver, manager, or protocol code that meets your needs. To do this, create an AIX Operating System file for the actual code. The module must conform to the programming restrictions described in "Developing and Adding Code to the VRM" on page 2-27. In particular, the module must be designed to be memory location-independent.

2. Compile the module with the AIX Operating System **vcc** command.

   You can compile one or more modules with the **vcc** command. Do not try to use another compile command (such as **cc**) because the subsequent conversion will not complete successfully.

   The **vcc** command is issued as follows:

   ```
   vcc  file1 <file2>
   ```

   This AIX Operating System command compiles your program source code and produces an **a.out** format object module. This **a.out** module will be input to the **vrmfmt** command. The **vrmfmt** command converts the **a.out** module to a VRM-compatible object module.

3. Convert the output from **vcc** with the AIX Operating System **vrmfmt** command.

   After successfully compiling a module with **vcc**, you must convert it to VRM-executable object module format. The **vrmfmt** command does this conversion and is issued as follows:

   ```
   vrmfmt input_file <output_file>
   ```

   In the preceding example, `input_file` is the name of the module produced by **vcc**. As shown by `output_file`, you can optionally specify a name for the converted module. If you do not specify a name for the converted module, the module is placed into a file called **a.vrm**. If you perform successive conversions without specifying unique output file names, you will get a message stating that the file name (**a.vrm**) already exists. At this point, you can write over the existing module in **a.vrm** or restart the conversion, specifying a unique output file name.

4. Revise the AIX Operating System configuration files.

   IBM RT PC AIX Operating System Technical Reference describes the AIX Operating System files that you need to revise in order to configure your added code at each operating system IPL. If you do not revise these files, your added code will not be configured with the VRM at the next IPL.

   If you do not want your code to be configured as part of the system at every IPL, do not revise the AIX Operating System configuration files. Instead, simply follow the preceding instructions regarding development of the code and the use of the AIX Operating System **vcc** and **vrmfmt** commands. Then you must create a DDS for your module, explicitly issue the **Define Code SVC** (to install your new code module onto the VRM minidisk) and the **Define Device SVC** (to send the driver's DDS into the VRM address space). If you do not intend to configure the module at each operating system IPL, ignore the next step.

5. Issue the AIX Operating System **vrmconfig** command.

   At this point, you have coded, compiled, and converted a C language module to VRM-executable format. Now you must install the code into VRM address space.

   To do this, issue the **vrmconfig** command with the appropriate flags. Refer to *IBM RT PC AIX Operating System Commands Reference* for a description of the flags available for use with **vrmconfig**.

   You can also perform a re-IPL of the system to move the new code to the VRM minidisk. A **vrmconfig** command is issued automatically during each IPL. However, a total re-IPL has the following disadvantages when compared to direct execution of **vrmconfig** from a running system:

   - A re-IPL takes more time than direct execution of **vrmconfig**.
   - Any errors you get while adding the code are written to a file during re-IPL. When you issue **vrmconfig** from a running system, any error messages you get are sent to the display screen.

Chapter 3. Virtual Terminal Subsystem

CONTENTS

About This Chapter

This chapter documents the interface between the virtual machine and the virtual terminal manager (VTM). The VTM is comprised of the virtual terminal resource manager (VTRM), a virtual terminal mode processor (VTMP), and device drivers. Each of these components has a separate command interface to the virtual machine. The command interfaces and information returned to the operating system from the components are described in the following sections.

This chapter also includes a C language example of a device-specific module for a display device. This type of module is required when you add code to the system to support a new display type.

Virtual Terminal Manager

The virtual terminal manager is a collection of VRM components which extend the function of interactive input/output hardware. The virtual terminal manager controls the physical terminal, a collection of output devices (displays, speaker) and input devices (keyboard, locator), and maps virtual terminals to virtual machines.

Each instance of a physical terminal requires a VTM. VTM components include:

- Virtual Terminal Resource Manager (VTRM) — This virtual device manager coordinates the actions of all virtual terminals. The VTRM consists of two parts, a resource controller and a screen manager.

  The resource controller handles virtual terminal configuration, initialization, opening and closing. The screen manager handles real input routing from the keyboard or locator and coordinates the use of display screens and the speaker.

- Virtual Terminal Mode Processor (VTMP) — An instance of this device manager exists for each instance of a virtual terminal. The VTMP controls operation modes, datastream information, and other display data format and routing considerations.

- Device drivers — Each instance of a virtual terminal requires one instance of a virtual display driver. One keyboard device driver, however, is sufficient for use by all virtual terminals. The VTM can also support optional devices, such as a locator device, speaker, and multiple display devices.

The VTMP is an example of a device manager model. A device manager coordinates the actions of multiple device drivers (and possibly of multiple processes). Typically, the operating system can send commands directly to the device driver controlling a hardware adapter. In the case of the VTM, however, the operating system is restricted from commanding the device drivers directly. Figure 3-1 on page 3-5 shows the SVC variations directed to the VTM. You may want to refer back to this illustration when you read "Virtual Machine Interface to the VTRM" on page 3-6 and "Virtual Machine Interface to the VTMP" on page 3-26.

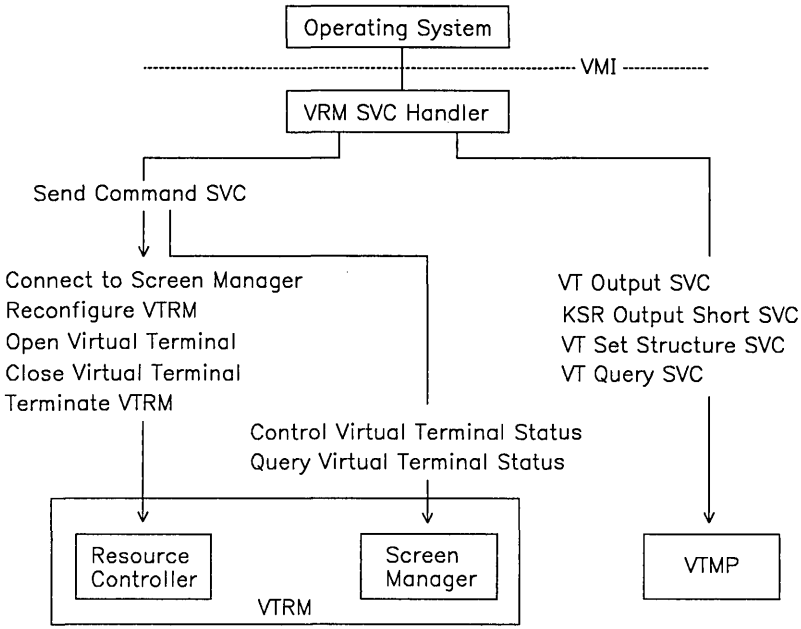


Figure 3-1. SVCs Directed to Virtual Terminal Components

In the case of the VTM, the operating system communicates with SVCs across the VMI to the VTRM and VTMP. These VRM processes then enqueue work requests to the appropriate processes or hardware device drivers. In addition, the VTMP can make direct calls to device-specific code bound to it.

The operating system sends two basic types of SVCs to VTM components. The **Send Command SVC**, directed to the VTRM, handles virtual terminal resource requests and virtual terminal status operations. By changing the options field of the **Send Command SVC**, you can use any of the commands to the resource controller or screen manager.

The VTMP SVCs (such as **VT Set Structure SVC** and **VT Output SVC**) control the format and routing of display data.

Virtual Machine Interface to the VTRM

The VTRM presents two distinct interfaces to a virtual machine. Both interfaces follow the programming conventions defined by the VRM input/output subsystem. The virtual machine communicates requests to the VTRM through **Send Command SVCs**. After servicing a request, the VTRM returns results either by a synchronous return from the SVC or by a virtual interrupt, whichever the virtual machine requests.

The first interface presents the functions of a device manager for resource control. Control functions include opening and closing virtual terminals, querying or modifying the configuration of the physical terminal, and so on. The resource control interface takes the form of queued **Send Command SVCs** and is known as QRC (queue for resource control). See Figure 3-2 on page 3-7.

The virtual machine establishes the resource control interface by defining (if necessary) and attaching to the VTRM. The virtual machine directs resource control request SVCs to the IODN for the resource control interface. The VRM SVC handler queues the requests to the resource controller of the VTRM.

Send Command SVC variations directed to the resource controller of the VTRM include:

- Reconfigure VTRM
- Open Virtual Terminal
- Close Virtual Terminal
- Connect to Screen Manager
- Terminate VTRM.

The second interface between the VTRM and the VMI presents a specialized screen manager, which provides such functions as hiding or activating virtual terminals. This interface is established when the resource controller attaches the screen manager process to the virtual machine. The screen management interface is also comprised of **Send Command SVCs** and is known as QSM (queue for screen management). See Figure 3-2 on page 3-7. The virtual machine directs screen management request SVCs to the IODN for the screen management interface. The VRM SVC handler queues the requests to the screen manager of the VTRM.

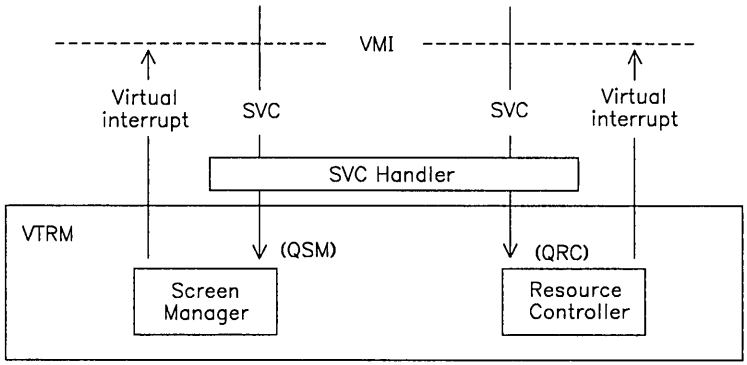


Figure 3-2. Virtual Machine Interface to VTRM Components. For both resource control and screen management, the interface to the VMI consists of SVC commands issued by the virtual machine and virtual interrupts generated by the VTRM (or synchronous returns from the SVCs).

Send Command SVC variations directed to the screen manager component of the VTRM include:

- Control Virtual Terminal Status
- Query Virtual Terminal Status

All SVC requests to the VTRM (five resource control functions and two screen management functions) use the **Send Command SVC** with a different operation option. The operation option is reflected in bits 20-31 of calling register GPR2.

Before a virtual machine can issue a **Send Command SVC** to the VTRM, the virtual machine must attach to the VTRM with the **Attach Device SVC**. A return parameter from the attach is a path identifier (path ID). The path ID determines to which component a command is sent.

The resource controller is also assigned an IODN. The first instance of the resource controller has an IODN of 0x0205. The screen manager's IODN is assigned by the VRM when the resource controller creates the screen manager interface to the virtual machine. The resource controller returns the screen manager's IODN and path ID to the virtual machine in response to the **Connect to Screen Manager** request.

The **Send Command SVC** variations directed to the resource controller and screen manager are defined on the following pages. Refer to Figure 3-1 on page 3-5 to see which component handles which SVC.

Reconfigure Virtual Terminal Manager

Description: The virtual machine issues a **Send Command SVC** with the reconfigure device option to change the configuration of the physical terminal or to change the virtual terminal defaults. For example, you can have up to four display devices, one locator, one speaker, and up to 32 fonts associated with a physical terminal.

Calling Register Conventions:

GPR2 =

Bits 0-15 =	Resource controller IODN
Bit 16 =	Interrupt on completion *
Bit 17 =	Interrupt on error *
Bit 18 =	Synchronous operation *
Bit 19 =	0 for no command extension
Bits 20-31 =	0 for reconfigure devices option.
	* indicates caller's option

GPR3 = Requested operation

Reconfigure operations 0 through 4 and 10 through 16 will be in effect only for terminals opened subsequent to the reconfigure. Operations 5 through 9 and 18 will be in effect for terminals currently open as well as those opened subsequent to the reconfigure.

0 = Add a physical locator
1 = Add a physical sound device
2 = Add a physical display
3 = Delete a physical display
4 = Add a font
5 = Change keyboard typamatic rate
6 = Change keyboard typamatic delay
7 = Change locator sample rate
8 = Turn keyboard click mechanism on or off
9 = Set sound volume level
10 = Replace position code map
11 = Replace character code map
12 = Replace echo/break map
13 = Replace miscellaneous default values
14 = Set the default display for newly-opened virtual terminals.
15 = Add physical dials
16 = Add physical lighted programmable function keys
18 = Change DMA start address and length

GPR4 = Object of the operation

Operation 0 Physical locator device driver IODN

Operation 1 Physical sound device driver IODN
Operation 2 Physical display identifier
Operation 3 Physical display identifier
Operation 4 Font IOCN
Operation 5 Keyboard-device dependent

Bits 24-31 of GPR4 indicate the keyboard typamatic rate. For the standard RT PC keyboard, valid values fall in the range 2 to 40 characters per second and can be changed in units of 1 character per second. The default value for the RT PC keyboard is 14 characters per second.

Operation 6 Keyboard-device dependent

Bits 16-31 of GPR4 indicate the keyboard typamatic delay. For the standard RT PC keyboard, valid values fall in the range 300 to 600 milliseconds and can be incremented in units of 100 milliseconds. The default value for the RT PC keyboard is 500 milliseconds.

Operation 7 Locator-device dependent

Bits 24-31 of GPR4 indicate the locator samples-per-second rate. For the standard RT PC locator, valid values are 10, 20, 40, 60, 80, or 100 samples per second. The default value for RT PC locators is 60 samples per second.

Operation 8 Keyboard-device dependent

Bit 31 of GPR4 indicates whether the speaker emits a click when a key is pressed. Sound is suppressed when bit 31 equals zero and emitted when bit 31 equals one. The default for the RT PC keyboard is click on.

Operation 9 Speaker-device dependent

Bits 24-31 of GPR4 indicate the volume of sounds emitted by the speaker. For the standard RT PC speaker, valid values are 0 (sound off), 1 (low volume), 2 (medium volume) and 3 (high volume). The default for the RT PC speaker is medium volume.

Operations 10 through 13 allow you to change certain system defaults by modifying some supplied files. These files are included with the HFT Examples Programs on the AIX Multi-User Services diskettes and are installed in the **/usr/lib/samples/hft** directory. Install the example programs and modify the specified file before performing reconfigure operations 10, 11, 12 or 13. The modified file you provide will be in effect for all virtual terminals after successful completion of the reconfigure command.

Operation 10 Keyboard translate table IOCN

After installing the HFT Example Programs, modify the file **hftkbdmap.c** to the desired mapping, run it through the TOC converter with the **vrmcc** and **vrmfmt** commands, issue a

Define Code SVC, then perform the **Send Command SVC** with the reconfigure virtual terminal manager option.

See **Define Code SVC** in *VRM Programming Reference* for valid IOCNs for the table.

Operation 11 Character code map IOCN

The character code map is a structure that includes both the Unique 1 (:) and Unique 2 (;) display maps.

After installing the HFT Example Programs, modify the file **hftchrmap.c** to the display map you want, run it through the TOC converter with the **vrmcc** and **vrmfmt** commands, issue a **Define Code SVC**, then perform the **Send Command SVC** with the reconfigure virtual terminal manager option.

See **Define Code SVC** in *VRM Programming Reference* for a valid IOCN for the new map.

Operation 12 New echo/break map IOCN

After installing the HFT Example Programs, modify the file **hftecbrmap.c** to the echo/break map you want, run it through the TOC converter with the **vrmcc** and **vrmfmt** commands, issue a **Define Code SVC**, then perform the **Send Command SVC** with the reconfigure virtual terminal manager option.

See **Define Code SVC** in *VRM Programming Reference* for a valid IOCN for the new map.

The default structure is initialized so that all echo bits are set and all break bits are not set.

Operation 13 Miscellaneous defaults IOCN

After installing the HFT Example Programs, modify the file **hftmiscdef.c** to the defaults you want, run it through the TOC converter with the **vrmcc** and **vrmfmt** commands, issue a **Define Code SVC**, then perform the **Send Command SVC** with the reconfigure virtual terminal manager option.

See **Define Code SVC** in *VRM Programming Reference* for a valid IOCN for the new structure.

Operation 14 Physical display identifier.
Operation 15 Physical dials device driver IODN (bits 16-31)
Operation 16 Physical lighted programmable function keys device driver IODN (bits 16-31).
Operation 18 Start address of DMA area.

GPR5 = Additional information required for operations 2 and 18:

- Operation 2:

  Bits 0-15 = Display device driver IODN
  Bits 16-31 = Device-specific module IOCN

- Operation 18: GPR5 contains the length of the DMA area.

GPR6 = This is a user field set up by the operating system device driver and returned by the VTRM in a PSB.

GPR7 = Path identifier

Return Codes:

The following error codes are returned in GPR2 (for synchronous operations) or in the Operations Results field of the PSB if the resource controller detects errors while processing the SVC:

6405 = Specified device is already part of the physical terminal configuration
6406 = Table full
6407 = Device in use
6408 = Invalid device identifier
6409 = Device not present
6415 = Change rejected by device
6416 = Invalid dials device driver IOCN
6426 = Invalid LPF keys device driver IOCN
6431 = Invalid locator device driver IODN
6432 = Invalid locator device driver IOCN
6433 = Invalid locator device query
6434 = Locator device driver incorrectly bound
6436 = Invalid dials device driver IODN
6441 = Invalid display device driver IODN
6444 = Virtual display driver bind error
6446 = Invalid LPF keys device driver IODN
6451 = Invalid sound device driver IODN
6452 = Invalid virtual display driver IOCN

6470 = Resources unavailable
6471 = Operation completed, cannot recover resources
6472 = Invalid font IOCN
6473 = Font module address error
6474 = Font bind error
6480 = Invalid operation
6482 = Invalid virtual terminal default IOCN
6490 = Invalid device option
6499 = Unsuccessful VTRM initialization.

The following figure shows the PSB returned from the reconfigure request.

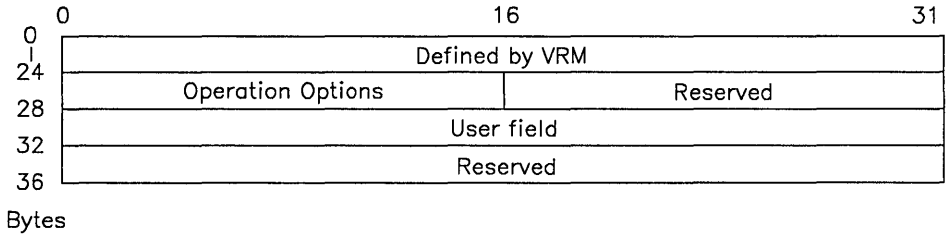


Figure 3-3. Reconfigure Virtual Terminal Manager PSB

Comments: The resource controller cannot verify the IODNs and IOCNs received with this SVC. If the received IODN or IOCN is known to the VRM, but does not conform to the expected component type, the results are unpredictable.

Also, the resource controller does not delete any devices or code modules rendered useless to the VTM by the reconfigure command. The virtual machine must delete these resources, if desired.

You can find the VTRM's current device characteristics by issuing a **Query Device SVC** to the VTRM's IODN. One of the return parameters is the current physical terminal configuration. For details on the specifics of the device characteristics fields, see "Virtual Terminal Resource Manager" on page 3-202.

If the resource controller detects an error during a reconfiguration request, no changes are made.

Open a Virtual Terminal

Description: The resource controller initializes the data structures, allocates the system resources, and creates the process required for an instance of a virtual terminal. The resource controller attaches the virtual terminal to the virtual machine. The new virtual terminal's IODN and path ID are then returned by the resource controller to the virtual machine.

Calling Register Conventions:

GPR2 =

Bits 0-15 =	Resource controller IODN
Bit 16 =	Interrupt on completion *
Bit 17 =	Interrupt on error *
Bit 18 =	Synchronous operation *
Bit 19 =	0 for no command extension
Bits 20-31 =	2 for open virtual terminal.
	* indicates caller's option.

GPR3 = Not used

GPR4 = Interrupt information

Bits 0-4 =	Not used
Bits 5-7 =	Solicited interrupt level
Bits 8-15 =	Solicited interrupt sublevel
Bits 16-20 =	Not used
Bits 21-23 =	Unsolicited interrupt level
Bits 24-31 =	Unsolicited interrupt sublevel

GPR5 = VTMP IOCN

Bits 0-15 =	Set equal to zero
Bits 16-31 =	IOCN of VTMP, or 0 for default.

GPR6 = This is a user field set up by the operating system device driver and returned by the VTRM in a PSB.

GPR7 = Path identifier

Return Codes: contained in GPR2

The following error codes are returned in GPR2 or in the Operation Results field of the PSB when the resource controller detects errors while processing the SVC:

6402 = Invalid VTMP IOCN
6450 = Invalid solicited interrupt level
6454 = VTMP module unsuccessfully bound to resource controller

6460 = Invalid unsolicited interrupt level
6461 = Maximum number of virtual terminals open
6470 = Resources unavailable, no virtual terminal opened
6490 = Invalid device option
6499 = Unsuccessful VTRM initialization.

The following figure shows the returned PSB of the open virtual terminal.

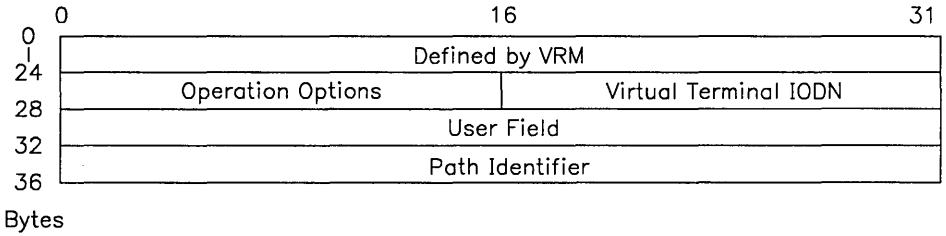


Figure 3-4. Open Virtual Terminal PSB

Comments: The IOCN of the VTMP supplied by the **Send Command SVC** with open virtual terminal options must identify a code module that meets the VTMP interface specifications. The resource controller does not make a copy of the supplied module; the virtual machine must do this, if desired.

The virtual machine must attach to the resource controller before issuing the open virtual terminal request.

The interrupt sublevels cannot be checked.

The screen manager activates the newly opened virtual terminal (unless the currently active virtual terminal cannot be deactivated).

Close a Virtual Terminal

Description: Upon receiving a close request from the virtual machine, the resource controller sends a queue element to the screen manager that identifies the virtual terminal to close. When the screen manager acknowledges that the terminal is closed, the resource controller frees the system resources and clears the data structures initialized when the terminal was opened. If the target virtual terminal is active, the screen manager activates another terminal.

Calling Register Conventions:

GPR2 =

Bits 0-15	=	Resource controller IODN
Bit 16	=	Interrupt on completion *
Bit 17	=	Interrupt on error *
Bit 18	=	Synchronous operation *
Bit 19	=	0 for no command extension
Bits 20-31	=	3 for close virtual terminal

* indicates caller's option.

GPR3,4 = Not used

GPR5 =

Bits 0-15	=	Not used (0)
Bits 16-31	=	Virtual terminal IODN

GPR6 = This is a user field set up by the operating system device driver and returned by the VTRM in a PSB.

GPR7 = Path identifier.

Return Codes:

The following error codes are returned in GPR2 or in the Operation Results field of the PSB when the resource controller detects errors while processing the SVC:

6401 = Invalid virtual terminal IODN
6470 = Resources unavailable, cannot close virtual terminal
6471 = Cannot recover resources, virtual terminal closed
6490 = Invalid device option
6499 = Unsuccessful VTRM initialization.

Figure 3-5 on page 3-16 shows the PSB returned for the close request.

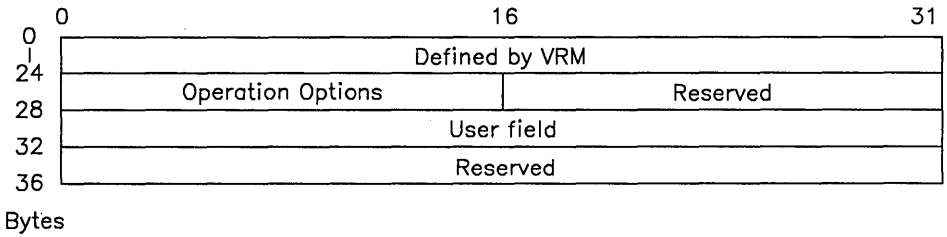


Figure 3-5. Close Virtual Terminal PSB

Comments: The **Send Command SVC** with close virtual terminal option can be issued only after the virtual machine has attached to the VTRM.

Connect to Screen Manager

Description: This command causes the resource controller to attach the screen manager to the virtual machine. The resource controller then returns the screen manager IODN and the ID of the path from the screen manager to the virtual machine.

The screen manager returns both solicited and unsolicited interrupts on this path. Solicited interrupts occur as a result of SVCs to the screen manager. An unsolicited interrupt is sent to a virtual machine when its last (or only) virtual terminal is closed. In this case, the pertinent fields in the interrupt structure are:

- Operation results field = 6489
- IODN = screen manager IODN.

Note that this unsolicited interrupt is not sent if the virtual machine issues a **Detach SVC** from the screen manager.

Calling Register Conventions:

GPR2 =

Bits 0-15 =	Resource controller IODN
Bit 16 =	Interrupt on completion *
Bit 17 =	Interrupt on error *
Bit 18 =	Synchronous operation *
Bit 19 =	0 for no command extension
Bits 20-31 =	10 for connect to screen manager

* indicates caller's option.

GPR3 = Not used

GPR4 = Interrupt information

Bits 0-4 =	Not used
Bits 5-7 =	Solicited interrupt level
Bits 8-15 =	Solicited interrupt sublevel
Bits 16-20 =	Not used
Bits 21-23 =	Unsolicited interrupt level
Bits 24-31 =	Unsolicited interrupt sublevel

GPR5 = Not used.

GPR6 = This is a user field set up by the operating system device driver and returned by the VTRM in a PSB.

GPR7 = Path identifier.

Return Codes:

The following error codes are returned in GPR2 or in the Operation Results field of the PSB when the resource controller detects errors while processing the SVC:

6450 = Invalid solicited interrupt level
6460 = Invalid unsolicited interrupt level
6470 = Resources unavailable, screen manager not connected
6481 = Too many virtual machines attached
6490 = Invalid device option
6491 = Screen manager already connected
6499 = Unsuccessful VTRM initialization.

The following figure shows the PSB returned for a connect to screen manager request.

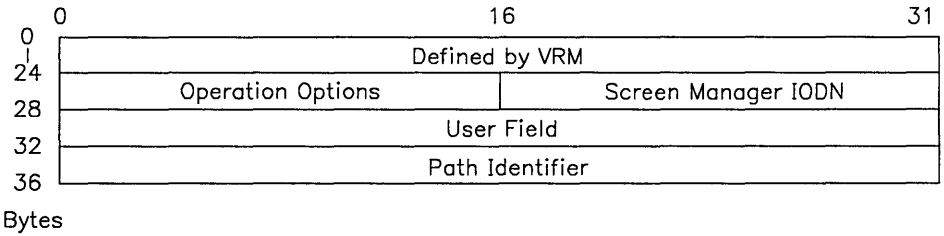


Figure 3-6. Connect to Screen Manager PSB

Comments: As many as 32 virtual machines may be attached to the screen manager at the same time.

This SVC can be issued only after the virtual machine has attached to the resource controller.

If the virtual machine detaches from the screen manager (with the **Detach Device SVC**), the screen manager cannot generate an unsolicited interrupt when the virtual machine's last terminal is closed.

Terminate VTRM

Description: A virtual machine can temporarily or permanently terminate the VTRM and virtual terminal management with this SVC.

Note: If termination of IBM-supplied virtual terminal management components is to be temporary, you should issue a **Query Device SVC** with the VTRM's IODN *before* you issue the terminate request. This SVC returns the VTRM's define device structure (DDS). You can restart the VTM by issuing two **Define Device SVCs**. Issue the first SVC with the 'delete' option to delete the IBM-supplied IODN for the VTRM (0x0205). Issue the second SVC with the 'add' option and define a new, unused IODN to reinstate the VTM.

The resource controller will not terminate the VTRM if it detects any open virtual terminals. If the resource controller determines that all virtual terminals are closed, termination begins. First, the resource controller terminates the screen manager with the VRM **_signal** routine. The resource controller releases any system resources used by the screen manager, then returns, terminating itself. The resource controller cannot delete the device drivers or any code modules, including its own code module. The virtual machine must make any of these deletions, if desired.

After you terminate the VTRM device manager processes, only the device drivers remain in the VRM. However, because these drivers cannot be attached to any virtual machine, none can be activated. You must define and attach your own drivers for use with the I/O devices previously under VTM control if you require VRM device driver services for interrupts or direct memory access.

Calling Register Conventions:

GPR2 =

Bits 0-15	=	Resource controller IODN
Bit 16	=	Interrupt on completion *
Bit 17	=	Interrupt on error *
Bit 18	=	Synchronous operation *
Bit 19	=	0 for no command extension
Bits 20-31	=	9 for terminate VTRM
		* indicates caller's option.

GPR3,4,5 = Not used

GPR6 = This is a user field set up by the operating system device driver and returned by the VTRM in a PSB.

GPR7 = Path identifier.

Return Codes:

The following error codes are returned in GPR2 and the Operation Results field of the PSB when the resource controller detects errors while processing the SVC:

6471 = VTRM terminated, but cannot recover all resources
6490 = Invalid device option
6497 = Virtual terminal open, cannot terminate VTRM.

The following figure shows the PSB returned for a terminate VTRM request.

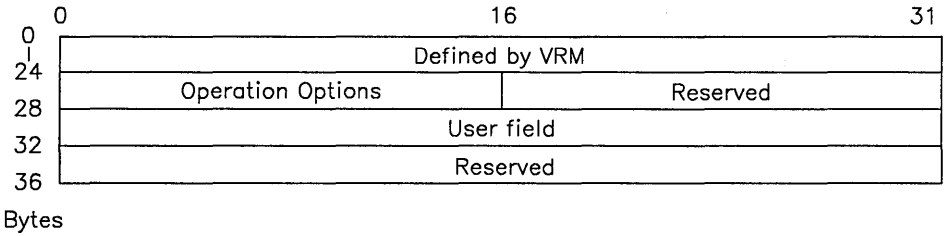


Figure 3-7. Terminate VTRM Request PSB

Control Virtual Terminal Status

Description: The virtual machine issues this command to the screen manager. This command basically controls the status of virtual terminals available to a virtual machine. All the virtual terminals owned by all the virtual machines are linked together in a group called the screen manager ring. The screen manager places an entry in the ring for each virtual terminal opened. The terminal that is currently active is called the "head" of the ring; the last terminal on the ring is called the "tail." When a new terminal is added to the ring, the terminal becomes the head of the ring.

Two multi-key sequences allow you to switch between virtual terminals and control which terminal is currently active. The active terminal is defined as the terminal that will accept keyboard or locator input and will update the physical display. When you press the **Alt + Action** keys, the screen manager makes the next virtual terminal active. This relationship is indicated by "a" in Figure 3-8. When you press **SH + Action**, the screen manager makes the last virtual terminal active. This relationship is indicated by "b" in Figure 3-8.

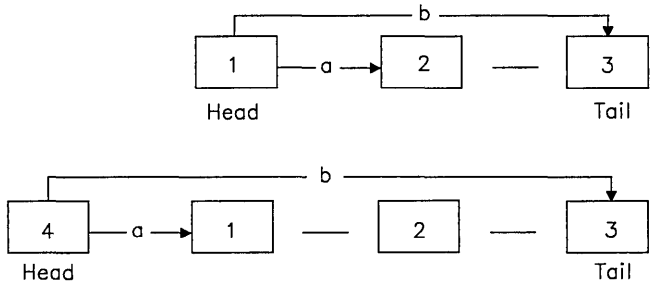


Figure 3-8. Screen Manager Ring Examples. In these figures, 'a' indicates the path from the active virtual terminal to the next virtual terminal and 'b' indicates the path from the active virtual terminal to the last virtual terminal.

Note that when you have three entries in the ring, you can access all the terminals with a single key sequence. With four or more entries, you have to skip from one terminal to another in some cases to activate a particular terminal. For example, in the preceding figure with four terminal entries, terminal #2 cannot be accessed from the active terminal #4 without first skipping to terminal #1 or terminal #3.

You can remove a terminal from the virtual machine's working set by using the hide option of this command. When a terminal is hidden, it is passed over when its place in the ring would ordinarily make it the active terminal.

The following options are provided with this command:

- **Activate**

  The activate command places a virtual terminal (specified by IODN) at the head of the screen manager ring, and thus makes the terminal the active terminal. This option also clears the terminal's hidden flag. The screen manager cannot activate the virtual terminal if the current active virtual terminal cannot be deactivated.

- **Hide**

  The hide command sets a flag in the terminal's ring entry indicating that the screen manager should not activate this terminal (identified by IODN). This does not affect the terminal's position in the ring. From the time the hidden flag is set, the screen manager ignores the terminal's presence in the ring until an unhide command is issued. If the virtual terminal was active when the hide command was issued, the screen manager makes the terminal inactive (if possible) but does not prevent the virtual machine attached to the virtual terminal from communicating with the virtual terminal. Hiding the active virtual terminal has the same effect as the last terminal function. If the virtual terminal hidden is the last one hidden in the ring, the contents of the physical display screen remain the same; no component attempts to clear the screen.

- **Unhide**

  The unhide command resets the hidden flag. If a virtual terminal was hidden, the unhide command restores the presence of the terminal in the ring, but does not affect the terminal's ring position or make the terminal active. If the hidden virtual terminal happens to be at the head of the ring when the **unhide** command is issued, that terminal becomes visible and, in a sense, active.

- **Set Command Virtual Terminal**

  The Set Command Virtual Terminal option identifies to the screen manager which terminal to treat as the command virtual terminal. The only advantage to the command virtual terminal is that it is immediately activated when you enter the key sequence **Cntl + Action** or press both locator keys simultaneously with locator defaults enabled.

- **Enable Command Virtual Terminal from Locator**

  This option allows you to activate the command virtual terminal (if defined) by pressing both locator keys simultaneously. Because this is a default function of the locator, you do not need to issue this option unless you have previously disabled the function with the Disable Command Virtual Terminal option.

- **Disable Command Virtual Terminal from Locator**

  This option overrides the locator default that activates a command virtual terminal by pressing the locator keys simultaneously. After disabling the default locator function, you may define another locator function to the locator keys.

Calling Register Conventions:

GPR2 =

Bits 0-15 = Screen manager IODN.
Bit 16 = Interrupt on completion *
Bit 17 = Interrupt on error *
Bit 18 = Synchronous operation *
Bit 19 = 0 for no command extension
Bits 20-31 = 31 for Control Virtual Terminal Status
* indicates caller's option

GPR3 =

Bits 0-15 = Not used (0)
Bits 16-31 = IODN of the virtual terminal

GPR4 =

Bits 0-23 = Not used (0)
Bits 24-27 = Operation option
0 = Set Command Virtual Terminal
1 = Activate
2 = Hide
3 = Unhide
4 = Enable command VT activation from locator
5 = Disable command VT activation from locator
Bits 28-31 = Reserved

GPR5,6 = Not used

GPR7 = Path identifier.

Return Codes:

The following error codes are returned in GPR2 or in the Operation Results field of the PSB when the screen manager detects an error while processing the SVC:

6401 = Invalid virtual terminal IODN
6476 = Virtual terminal cannot be deactivated
6477 = Command virtual terminal designation already exists
6480 = Invalid operation
6490 = Invalid device option.

Comments: Only one command virtual terminal may be defined. This command terminal cannot be redefined unless the original command virtual terminal is closed.

Only unhidden virtual terminals are included in your working set of virtual terminals. Terminals in the working set can be activated using the keyboard or, for the commmand terminal only, by pressing both locator keys or the **Cntl + Action** key sequence.

Query Virtual Terminal Status

Description: The Query Virtual Terminal Status command allows you to determine whether an open virtual terminal is active or hidden. In addition, you can find out if a specified terminal is the command virtual terminal.

Calling Register Conventions:

GPR2 =

Bits 0-15	=	Screen Manager IODN.
Bit 16	=	Interrupt on completion *
Bit 17	=	Interrupt on error *
Bit 18	=	Synchronous operation *
Bit 19	=	1 for command extension
Bits 20-31	=	30 for Query Virtual Terminal Status
		* indicates caller's option

GPR3,4 = Not used

GPR5 = Address of area in which to return status data

GPR6 = Length of the returned data area.

Return Codes:

The following error codes are returned in GPR2 or in the Operation Results field of the PSB when the screen manager detects an error while processing the SVC:

6475 = Insufficient query response data
6490 = Invalid device option.

Return Data Area: The screen manager returns the status data at the address indicated in the command extension field of the **Start I/O SVC** program status block. The returned information is shown in Figure 3-9 and defined in the section that follows.

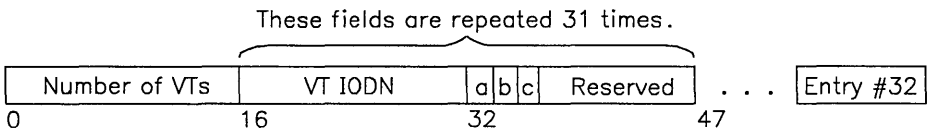


Figure 3-9. Status Data Returned by the Screen Manager

Number of VTs — This halfword indicates how many open virtual terminals are owned by the virtual machine issuing the query.

Bits 16 through 47 of this structure are repeated for each possible virtual terminal owned by the virtual machine, so you must reserve enough space for 32 such entries.

The 'Number of VTs' field tells you how many of the 32 entries are meaningful. The value of the subsequent bits is unpredictable.

VT IODN — This halfword is a virtual terminal IODN.

The next three bits provide additional status information about the virtual terminal.

When bit a = 1, the terminal is hidden.
When bit b = 1, the terminal is active.
When bit c = 1, the terminal is the command virtual terminal.

Virtual Machine Interface to the VTMP

The VTMP provides a model of a single terminal. In the case of multiple virtual terminals, each terminal must have a logically complete VTMP, whether that comes through shared data or copies of VTMP modules.

The VTMP operates in one of two modes. They are:

- Keyboard Send/Receive (KSR)
- Monitored Mode (MOM).

The KSR mode processor emulates an ASCII terminal and uses ANSI 3.64 controls, the RTASCII character set, and virtual terminal data structures. The VTMP defaults to KSR mode.

Monitored mode provides applications a direct output path to the display hardware and an optional shortened input path for the keyboard, locator, lighted programmable function (LPF) keys, dials, and display adapter.

Some forms of data that each mode accepts are unique to that mode. This design attempts to optimize the transfer of data between the VTMP and the application program and supports different functions within each mode.

Certain general functions to affect display, keyboard, locator, dials, and LPF key hardware, as well as sound generation, are available to either mode. With these you can:

- Turn keyboard LEDs on or off
- Set locator X and Y thresholds (for relative device) so locator inputs are reported only when thresholds are exceeded
- Set locator X and Y non-input area (for absolute device) so that locator inputs are reported only when the locator is within the input portion of the tablet
- Provide commands to generate sound on active and inactive terminals and cancel sound on the active terminal
- Set the LPF keys
- Set the granularities for reporting dials movement deltas
- Change the device on which a virtual terminal is displayed
- Return keyboard, display, locator, LPF key, and dials device characteristics.

For more information on these functions, see "General Major Data Type" on page 3-58.

The VTMP supplies default values for use of display, keyboard, locator, dials and LPF key hardware. Keyboard-to-character code mapping, character-to-display symbol mapping, echo specification, tab rack, and protocol mode flags are all provided by the VTMP. You can change these default values in two ways:

1. Define your own set of values for use with a specific virtual terminal with a **VT Set Structure SVC** or a **VT Output SVC**.
2. Change certain default values with a **Reconfigure VTRM SVC** (See "Reconfigure Virtual Terminal Manager" on page 3-8) and provide your own data structures.

The virtual machine interface to the VTMP consists of a set of SVCs and virtual interrupts. These include:

- **VT Output SVC**
- VT output acknowledge interrupt
- **KSR Output Short SVC**
- **VT Set Structure SVC**
- VT set structure acknowledge interrupt
- **VT Query SVC**
- VT query acknowledge interrupt
- VT unsolicited interrupts
- Acknowledge interrupt return codes.

Compared to the VTRM SVCs, which control the terminal resources, the SVCs to the VTMP deal primarily with display output. Virtual terminals receive data in GPRs 2-7 upon SVC execution. Register 2 contains the identifier of the path from the virtual machine to the virtual terminal. For all but the **KSR Output Short SVC**, one register contains the address of an output buffer for additional data. The **VT Query SVC** has two buffers, one for the query command and one for the query response.

The output buffer may contain RTASCII characters, ANSI 3.64 escape sequences, or virtual terminal data structures. Major data types are KSR, MOM and general (applies to both KSR and MOM). A virtual terminal data (VTD) ASCII control sequence (a private escape sequence) indicates the beginning of a data structure. A 32-bit length field indicates the end of a data structure, but a data structure may span multiple output buffers. The data between the end of a structure and the next VTD control is interpreted as ASCII codes.

Figure 3-10 on page 3-28 shows the data stream format.

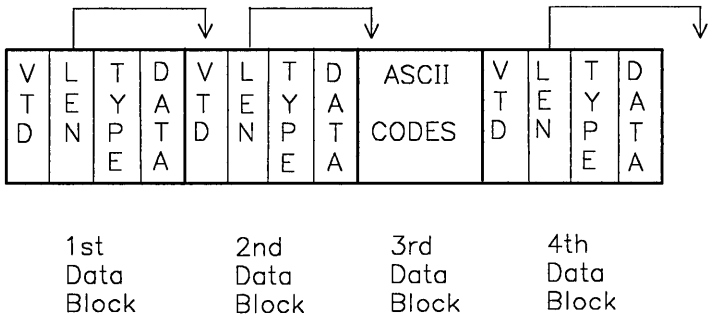


Figure 3-10. Data Stream Format

For more information on the characteristics and format of data structures, see "Major Data Types" on page 3-56.

The **VT Query SVC** uses the output buffer to return the requested query data from the VTMP. Interrupts return data in the old program status block (OPSB). The status field of the OPSB has bits set to distinguish between virtual interrupts and other interrupts, as well as solicited or unsolicited interrupts.

The following sections describe the SVCs directed to the VTMP and the interrupts returned to the virtual machine.

VT Output SVC

Description: You can send display data to the virtual terminal by way of the **VT Output SVC**. The data to be processed resides in a buffer in main storage. The virtual address of this buffer is passed in the SVC registers and will be converted to segment ID and segment offset for queueing. Returned information from this SVC indicates whether the SVC handler successfully queued this command for later processing by a virtual terminal. The VTMP generates an acknowledge interrupt in response to the **VT Output SVC**.

Note that the VTMP does not check the virtual machine-to-virtual terminal path ID parameter. However, the VRM checks this path and will return an error code if the path is invalid.

SVC Code: 0x**FFCC**

Calling Register Conventions:

GPR2 = Path identifier

This register contains a value that identifies the path from the virtual machine to the virtual terminal.

GPR3 = Virtual address of data buffer

GPR4 = Length of data buffer in bytes.

Return Codes: contained in GPR2

0 = Successfully queued

Comments: Since the output buffer contents, in storage, have not been queued, the virtual machine must refrain from modifying, deallocating or moving the buffer in storage until it receives an output acknowledge interrupt. Note also that the output buffer cannot span a virtual memory segment. Errors detected during output data processing can either be reported as a data stream item in an unsolicited interrupt, or in the return code of the acknowledge interrupt.

Virtual terminal output may sometimes come in the form of data structures. In this case, the beginning of a data structure is specified by the occurrence of a virtual terminal data (VTD) ASCII control sequence. The end of a data structure is specified by a length field. A data structure may span multiple output buffers. The data between the end of a structure and the next VTD control is interpreted as ASCII codes.

VT Output Acknowledge Interrupt

The output acknowledge interrupt tells you whether the data stream buffer contents were successfully processed from virtual machine storage. Figure 3-11 shows the contents of the old PSB.

A return code of zero indicates successful processing. "Acknowledge Interrupt Return Codes" on page 3-55 defines the unsuccessful return codes.

When you receive an output acknowledge interrupt, you know that the storage associated with the data buffer is now free for further manipulation or deallocation by the virtual machine.

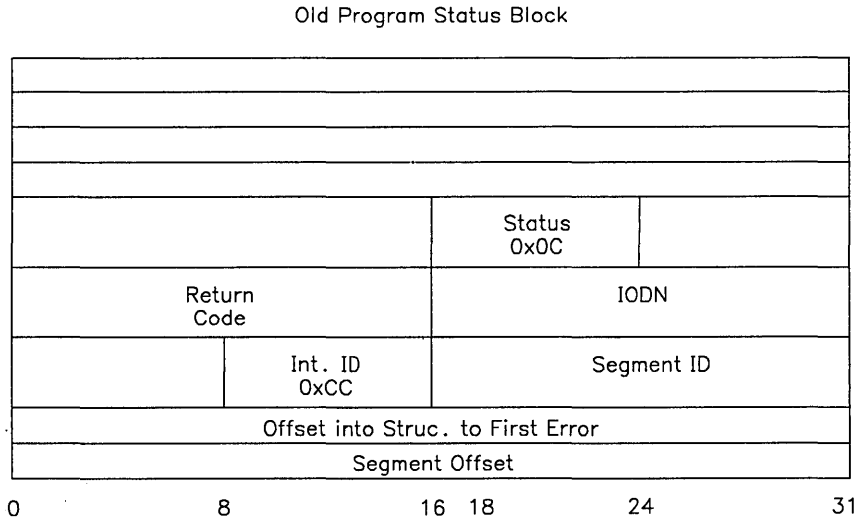


Figure 3-11. VT Output Acknowledge Interrupt

KSR Output Short SVC

Description: The **KSR Output Short SVC** allows passage of up to 16 bytes of KSR ASCII data to a single virtual terminal in registers without getting and maintaining storage buffers. Returned information from this SVC indicates whether a virtual terminal handler successfully queued this command for later processing. A **KSR Output Short SVC** generates no acknowledge interrupt.

SVC Code: 0x**FFCB**

Calling Register Conventions:

GPR2 = Path identifier

This register contains a value that identifies the path between the virtual machine and the virtual terminal.

GPR3 = (bits 0-7) Number of characters to transfer.

GPR4,5,6,7 = Character buffer.

Up to 16 bytes of data can be sent in these four GPRs as shown in the following chart.

KSR Output Short Registers

GPR4	Code 0	Code 1	Code 2	Code 3
GPR5	Code 4	Code 5	Code 6	Code 7
GPR6	Code 8	Code 9	Code 10	Code 11
GPR7	Code 12	Code 13	Code 14	Code 15
	0	8	16	24 31

Return Codes: contained in GPR2

0 = Successfully queued.

Comments: The data is interpreted as an ASCII data stream.

VT Set Structure SVC

Description: You can define the data in certain virtual terminal structures with the **VT Set Structure SVC**. Those structures include:

- Virtual terminal keyboard map
- Echo/break map.

The definition structure resides in a main storage buffer. The virtual address of the definition structure, passed in the registers, is converted to segment ID and segment offset for queueing to the VTMP.

Information returned from this SVC indicates whether the SVC was successfully queued for processing. The **VT Set Structure SVC** generates an acknowledge interrupt.

SVC Code: 0x**FFC8**

Calling Register Conventions:

GPR2 = Path identifier

This register contains a value indicating the path from the virtual machine to the virtual terminal.

GPR3 = User's virtual address pointer to structure

GPR4 = Length of the structure in bytes

GPR5 = (bits 0-7) Structure selector

2 = Keyboard mapping
3 = Echo/break mapping

Return Codes: contained in GPR2

0 = Successfully queued

Comments: Since only the register contents at the time of SVC execution are queued, ensure that the storage allocated for the structure is not modified, moved, or deallocated until the acknowledge interrupt is received. In addition, this buffer cannot span a virtual memory segment.

The **VT Set Structure SVC** may be issued by the virtual machine's device driver in response to certain out-of-data-stream commands.

Set Keyboard Map

The key position to code point mapping of a VTMP virtual terminal can be altered through the keyboard map structure.

The VTMP maintains a keyboard mapping table for each virtual terminal. This table maps the pressing of a key (as indicated by its key position, and whether the shift, control, alternate, or alternate graphics key is also held down) to a specific character, mode-processor-defined function, or string of characters. All or part of this mapping table can be modified by data passed in the VT KBD MAP structure.

The keyboard mapping structure contains a 2-byte header. This header is followed immediately by a set of key position mappings of variable length.

The keyboard mapping structure consists of three distinct mappings of its bytes, dependent on the type of key assignment. All key position mappings must specify a key position, the applicable shift state and the type of key assignment.

You can map four types of key assignments. Some of the types require different mapping specifications. The types and mapping requirements are:

(1) Single character and (2) single non-escaping character

- Key position
- Shift state
- Code page
- Code point.

(3) Function

- Key position
- Shift state
- 16-bit function ID.

(4) String

- Key position
- Shift state
- Code page
- Character string length
- Character string.

Functions are translated or executed according to the mode of the virtual terminal receiving input. The mapping types may be freely mixed but the number of mappings must match the number of key positions specified in Figure 3-12 on page 3-34.

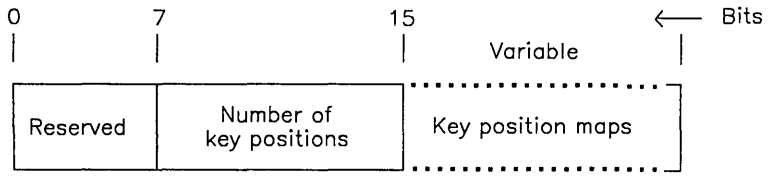


Figure 3-12. Keyboard Mapping Structure Header

The fields in the header pictured above are defined as follows:

- Number of key positions

  This value is the number of key positions to be mapped.

- Key position maps

  This field contains the new mappings of key position to symbol or function.

Figure 3-13 expands on the key position maps field from Figure 3-12.

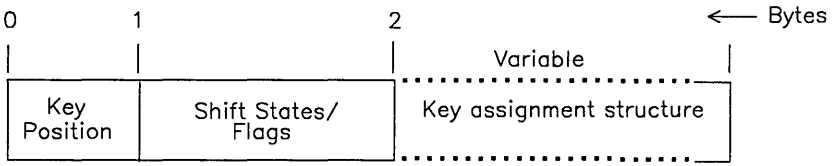


Figure 3-13. Keyboard Mapping Structure for Key Positions

The fields shown in the figure above are defined as follows:

- Key position

  The key position field contains the keyboard position to remap. Each key on the standard RT PC keyboard has a numeric position code that is used for this field. Figure 3-29 on page 3-106 matches the key to its position code.

- Shift states

  Bit 0 = caps lock (set this bit correctly regardless of the key state being mapped.)

  0 – Caps lock has no effect on this key.

  1 – Caps lock state in effect for this key.

  Bits 1-2

  The binary value of bits 1-2 indicates whether the new key assignment applies to the key in the base, shift, control, alternate, or alternate graphics state.

00 = base key (no shift, control, or alternate)
01 = shift plus base key
10 = control plus base key
11 = alternate plus base key.

Bit 3 — Alternate graphics

If bit 3 equals 1, bits 1 and 2 should be set to zero.

- Flags

  The binary value of bits 4-7 indicates the type of key assignment.

  0000 = A character is assigned to this key position. Bytes 2 and 3 contain a code page and code point for this field.

  0001 = A function is assigned to this key position. Bytes 2 and 3 contain a 16-bit internal ID.

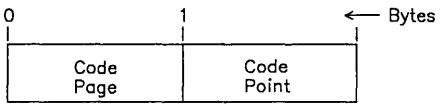
  0100 = A character string is assigned to this position. Byte 3 contains the length of the string (minus 1) in bytes. Although the maximum number of characters allowed is 255, the virtual machine has a maximum queue element depth of 15. Therefore, a maximum of 15 characters can be sent (assuming all 15 queue elements are available) in KSR mode or MOM mode if no ring buffer is defined. For MOM mode with a ring buffer defined, the maximum number of characters (255) can be accommodated if a ring buffer is defined accordingly. The total number of codes in all character strings cannot exceed 16K.

  0010 = A non-spacing character is assigned to this key position. Bytes 2 and 3 contain a code page and code point for this field.

- Key assignment structure

  A variable-length structure which contains key position assignments. The figures that follow describe the key assignment structure for the various flag values.

When the flags field = 0000 (character), the key assignment structure is defined as follows:

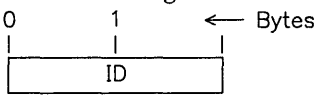


The fields shown in the preceding figure are defined as follows:

Code page The RTASCII character set from which the display symbol is taken.

Code point The RTASCII code point within the specified character set.

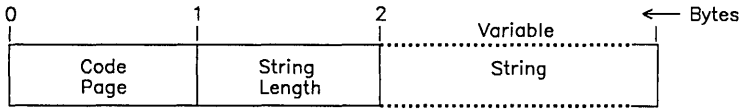
When the flags field = 0001 (function), the key assignment structure is defined as follows:



The field shown in the preceding figure is defined as follows:

ID A 16-bit value that identifies the function assigned to this key position. See "Valid Function Identifiers" on page 3-37 for a list of defined values.

When the flags field = 0100 (character string), the key assignment structure is defined as follows:



The fields shown in the preceding figure are defined as follows:

Code page RTASCII character set the display symbol is taken from.

String length An 8-bit value that indicates the length of the string (minus 1) in bytes. Although the maximum number of characters allowed is 255, the virtual machine has a maximum queue element depth of 15. Therefore, a maximum of 15 characters can be sent (assuming all 15 queue elements are available) in KSR mode or MOM mode if no ring buffer is defined. For MOM mode with a ring buffer defined, the maximum number of characters (255) can be accommodated if a ring buffer is defined accordingly.

String The character string to be assigned to the specified key position.

The following example uses the information provided in the preceding section. For this example, the key position 0x31 will be remapped to the character string 'EXAMPLE' in the P0 code page. The keyboard mapping structure would be coded as follows:

Byte	Value	Meaning
0	0	Reserved
1	1	Remap 1 key
2	0x31	Remap key position 0x31
3	0x04	Make the key a character string
4	0x3C	Use the P0 code page
5	0x07	Length of the string
6 - 12	0x45, 0x85, 0x41, 0x4D, 0x50, 0x4C, 0x45	the string 'EXAMPLE'

After the preceding reassignment, whenever the key at position 0x31 is pressed, the character string EXAMPLE is displayed.

Valid Function Identifiers

The VTMP recognizes certain function identifiers, and you can assign these functions to keys.

ID	Name
0x0000	Reserved
0x0100	Reserved
0x0101	CUU
	Move application cursor up one line
0x0102	CUD
	Move application cursor down one line
0x0103	CUF
	Move application cursor forward one character
0x0104	CUB
	Move application cursor backward one character
0x0105	CBT
	Move application cursor to the previous horizontal tab stop or beginning of field
0x0106	CHT
	Move application cursor to the next horizontal tab stop or beginning of field
0x0107	CVT
	Move application cursor down one vertical tab stop
0x0108	HOME
	Move application cursor to the first line, first character in presentation space
0x0109	LL
	Move application cursor to the last line, first character in presentation space
0x010A	END
	Move application cursor to the last line, last character in the presentation space
0x010B	CPL
	Move application cursor to the first character of the previous line
0x010C	CNL
	Move application cursor to the first character of the next line

0x0151 DCH

Delete the character over the application cursor

0x0152 IL

Insert one line following the line of the application cursor

0x0153 DL

Delete the line of the application cursor

0x0154 EEOL

Erase the characters from the application cursor to the end of the line

0x0155 EEOF

Erase the characters from the application cursor to the next tab stop

0x0156 CLEAR

Erase the entire presentation space

0x0157 INIT

Restores the virtual terminal to the state of a newly opened virtual terminal: erases all data in the presentation space, places the cursor at first position of first line, restores tab stops, modes, keyboard map, character map, and echo map to default values

0x0162 RI

Perform one line reverse index control

0x0163 IND

Perform one line index control

0x01FF Ignore

No information sent for this key.

Set Echo/Break Maps

"Echo" refers to sending the character associated with a keystroke to the display screen or performing the function associated with a control. "Break" is used in monitored mode (MOM) to switch the input path from the MOM input buffer to the unsolicited ASCII datastream flow. See "VT Unsolicited Interrupt" on page 3-45. Break map is not operational in KSR mode.

The echo and break mode for each ASCII graphic and control function is set or reset according to the echo/break structure. As with other mode settings, the provided default settings suffice for most situations. The default for KSR mode is to set echo for all graphics and controls.

Each virtual terminal maintains an echo table. When send/receive mode is reset, this table translates keyboard input to ASCII codes. Each ASCII graphic and control function has a corresponding echo bit.

In the echo map, a bit set to one means to execute the graphic or control functions resulting from keyboard translation as if the functions were received in the data stream from the host.

For example, if you key an 'a', and its echo bit is 1, the 'a' displays at the current cursor position of the display screen. An echo bit set to 0 indicates the keyed graphic or control is not echoed directly to the display until the information is received in an output data stream.

In the break map of a terminal in MOM mode, a bit set to one means to send keyboard input data through the normal unsolicited ASCII datastream flow. If the break bit is reset in MOM mode, the input data will be placed in the MOM input buffer ring if the ring was previously defined.

The echo/break structure consists of 32 consecutive words of storage aligned on a word boundary. The first 16 words make up the echo map and the last 16 words make up the break map. All 512 bits of each map have meaning. Each bit position from 0 to 255 inclusive specifies echo or break for the codes. Bit positions 256 to 511 specify echo or break for the extended control functions. These functions are shown in Figure 3-14 on page 3-40 and are sent as hexadecimal values. The control for **KSI**, for example, is 0x131. The 'INV' bit in each map indicates how to handle invalid or unsupported control sequences.

The echo/break structure is shared among code pages P0, P1, and P2. For P0 graphic code points, the echo/break bit tested is the ordinal of the code point 0x20 - 0xFF. For P1 and P2 code points, the bit tested is the ordinal of the code point with the high order bit always set. For example, the P0 code point 0x20 is tested against the 33rd bit in the map, but the P1 code point 0x20 is tested against the 161st bit in the map. All of the following code points are tested against the 161st bit:

P0 - 0xA0
P1 - 0x20
P1 - 0xA0
P2 - 0x20
P2 - 0xA0

Hex2	←Most Significant Hex Digits 0,1→ 10	11	12	13	14	. . .		1F
0	CBT	DMI		RC				
1	CHA	EMI	RI	KSI				
2	CHT	EA		INV				
3	CTC	ED	RIS					
4	CNL	EF	RM					
5	CPL	EL	SD					
6	CPR	ECH	SL					
7	CUB	GSM	SR					
8	CUD	HTS	SU					
9	CUF	HVP	SGR					
A	CUP	ICH	SG0					
B	CUU	IL	SG1					
C	CVT	IND	SM					
D	DCH	NEL	TBC					
E	DL	PFK	VTS					
F	DSR		SC					INV

Figure 3-14. Bit Positions of ASCII Controls in Echo Map. Note that bit positions 0x132 through 0x1FF are 'INV'.

The table in Figure 3-15 on page 3-41 shows the bit position numbering within the structure.

Words	Bits 0 31
0	0 31
1	32 63
2	64 95
3	96 127
4	128 159
5	160 191
6	192 223
7	224 255
8	256 287
9	288 319
10	320 351
11	352 383
12	384 415
13	416 447
14	448 479
15	480 511

Figure 3-15. Bit Position Numbering in Echo/Break Map

VT Set Structure Acknowledge Interrupt

The set structure acknowledge interrupt indicates that the new structure elements have passed from the virtual machine to the VRM. A successful update returns a zero return code. See "Acknowledge Interrupt Return Codes" on page 3-55 for definitions of unsuccessful return codes.

Old PSB word 6, bits 0-7, contains the structure selector value that identifies the structure. Receipt of the set structure acknowledge interrupt tells you that the storage associated with the structure is now free for manipulation and/or deallocation by the virtual machine.

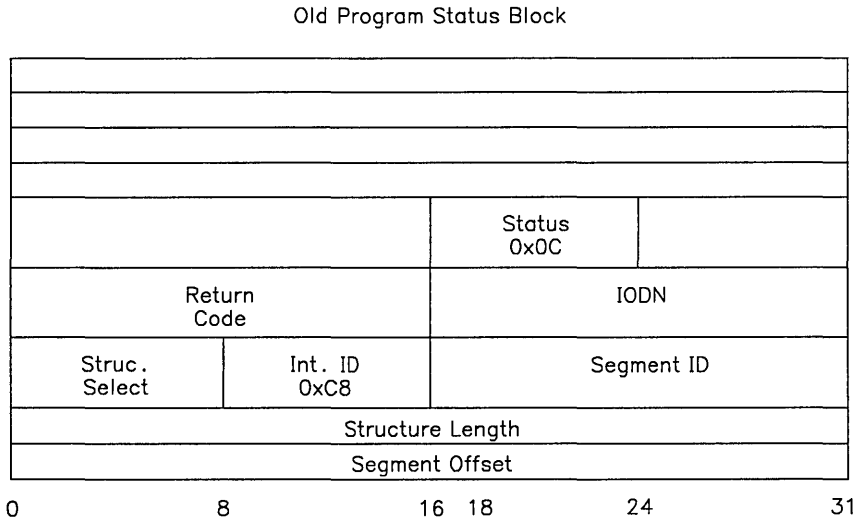


Figure 3-16. VT Set Structure Acknowledge Interrupt

VT Query SVC

Description: You can query (read) some VTMP data by issuing a **VT Query SVC**. The data to be queried is specified in the query command buffer, and the response is put in the query response buffer. The SVC parameter registers contain the addresses of both the command and response buffers. Information returned from this SVC indicates whether a query command was queued.

SVC Code: 0x**FFC6**

Calling Register Conventions:

GPR2 = Path identifier

This register contains a value which indicates the path between the virtual machine and the virtual terminal.

GPR3 = User's virtual address pointer to query command buffer

GPR4 = Length of query command buffer in bytes

GPR5 = User's virtual address pointer to query response buffer

GPR6 = Length of query response buffer in bytes.

Return Codes: contained in GPR2

0 = Successfully queued

Comments: A successful return notifies the device driver that the query SVC is queued for subsequent processing. A VT Query Acknowledge interrupt indicates completion of that processing. (See Figure 3-17 on page 3-44.)

The command and response data appear in VTD data block format as described for the VT output SVCs. However, minor type ranges for command, response, output, and interrupt data are unique.

Query response data will not overflow the designated buffer. If the response buffer is too small to return all of the query data, the application receives a return code of 6581 and the buffer contains as much query data as it can hold. To receive the complete query data, you should issue another **VT Query SVC** with a larger response buffer. Each query request should include only one query command. Data following the first query in the query command buffer is ignored.

This SVC can be issued by the virtual machine's device driver in response to certain out-of-data-stream commands.

Because only the register contents at the time of SVC execution are queued, the storage allocated for the structure cannot be modified, moved, or deallocated until reception of the query response interrupt. Storage for the structure must be allocated and initialized before SVC execution.

VT Query Acknowledge Interrupt

The query acknowledge interrupt indicates completion of a **VT Query SVC**. A return code of zero reflects a successful response; all unsuccessful return codes are defined in "Acknowledge Interrupt Return Codes" on page 3-55.

A return code of 6581 indicates that the response buffer was not large enough to return all the query response data. You should issue another **VT Query SVC** with a larger response buffer. In case of an error, the query command data length, segment offset, and segment ID fields of the OPSB will not be valid. The query command buffer length in the OPSB will be set to encompass the last byte of the query data. Command buffer processing stops when an error is detected. Figure 3-17 shows the format of the acknowledge interrupt.

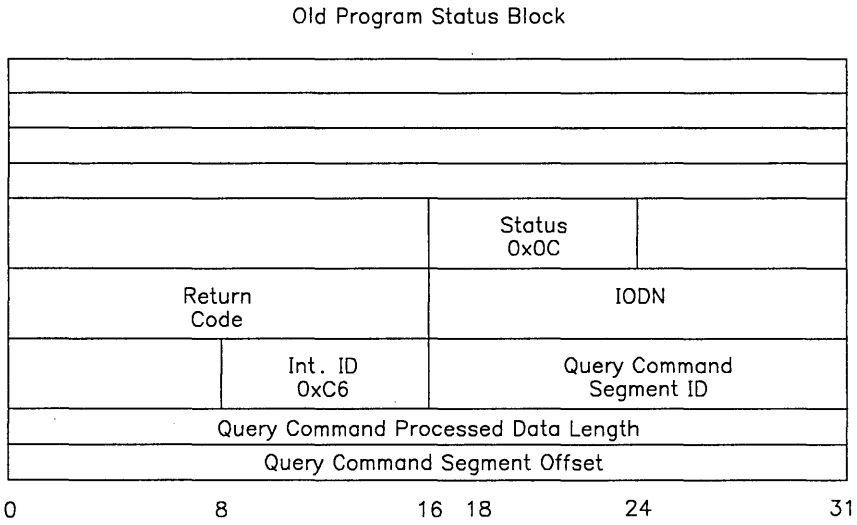


Figure 3-17. VT Query Acknowledge Interrupt

VT Unsolicited Interrupt

The virtual terminal sends an unsolicited interrupt to the virtual machine when the VTMP receives keyboard or locator input. Different interrupts return different amounts of information. Unsolicited interrupts return data directly to the old PSB.

A bit in the status flag byte of the old PSB distinguishes unsolicited interrupts from interrupts acknowledging completion of an SVC.

For unsolicited interrupts, the interrupt identifier field of the old PSB (word 6, bits 8-15) determines the dependent data fields in words 7 and 8. See Figure 3-18.

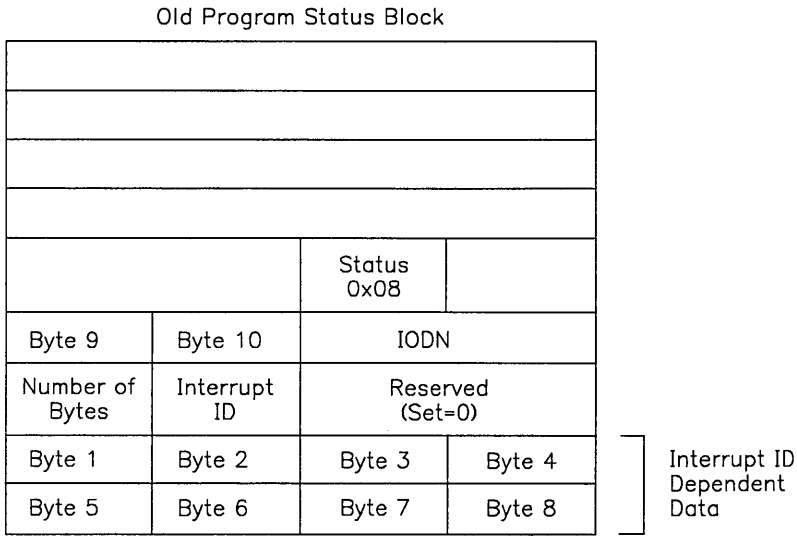


Figure 3-18. Unsolicited Interrupt Structure

In the preceding figure, the 'number of bytes' field indicates how many bytes of the 10-byte interrupt identifier-dependent data field are actually used.

The interrupt identifier field specifies a major and minor data stream type. The two high-order bits (bits 8 and 9) of word 6 determine the major type. The binary value of bits 8-9 determine the major data stream type as follows:

Bit Value	Major Type
00	Monitored
01	KSR

The six low-order bits (bits 10-15) of the interrupt identifier field determine which of 64 minor data types to associate with the unsolicited interrupt. A minor type value range from 0x80 to 0xBF is reserved in the virtual terminal data stream architecture for unsolicited interrupt data.

Unsolicited interrupts from input to KSR screens do not inhibit further input unless errors are reported. The returned data stream indicates errors in KSR data by way of device status report controls.

The following list defines interrupts according to minor type. Each definition notes variations due to major type. Interrupts 0-6 result from text cursor input. The returned interrupt type depends on data in the active terminal and protocol mode parameters set.

Where two ID values are specified, the actual value received depends on whether the terminal is in KSR or monitored mode.

ID = 40 ASCII data stream

The VTMP translates keystrokes into RTASCII data stream codes if XLATKBD mode is set. As many as 10 characters can be passed for each interrupt, so the number-of-data-bytes field must be examined at each interrupt to determine how many characters were passed for that interrupt.

OPSB Words				
5	Byte 9	Byte 10	Reserved	
6	# of bytes	Int. ID	Reserved (Set=0)	
7	Byte 1	Byte 2	Byte 3	Byte 4
8	Byte 5	Byte 6	Byte 7	Byte 8

ID = 04, 44 Key position, status, and scan code

This interrupt is returned when keyboard input is received with XLATKBD reset. The key position field identifies the logical key pressed. Key status bits indicate alternate, control, shift, caps lock, and num lock key states. The scan code and make/break bytes are hardware-dependent and require knowledge of the physical keyboard in use.

OPSB Words				Interrupt ID
7	Key Position	Scan Code	Key Status	04,44

Figure 3-19. Keyboard Data

The fields in the preceding figure are defined as follows:

- Key position

  This field reflects the position code unique to each key on the keyboard. Figure 3-29 on page 3-106 shows the values assigned to the standard RT PC keyboard.

- Scan code

  This field contains a hexadecimal value sent from the keyboard that determines how the keystroke is manifested.

- Key status

  This halfword indicates if any other keys are pressed with the reported keystroke. The bits are defined as follows:

  Bit 0 = Status of the shift key

  When this bit equals one, the key is pressed. When it is zero, the key is released.

  Bit 1 = Status of the control key

  When this bit equals one, the key is pressed. When it is zero, the key is released.

  Bit 2 = Status of the alternate key

  When this bit equals one, the key is pressed. When it is zero, the key is released.

  Bit 3 = Reserved

  Bit 4 = Status of the caps lock key

  When this bit equals one, the caps lock key is on. When the key is pressed again, the caps lock key is off. This also reflects the status of the keyboard LED.

  Bit 5 = Status of the num lock key

  When this bit equals one, the num lock key is on. When the key is pressed again, the num lock key is off. This also reflects the status of the keyboard LED.

  Bit 6 = Reserved

Bit 7 = Make/break status

1 = make, 0 = break.

Bit 8 = Repeat status

When this bit equals one, repeat is on. When it is zero, repeat is off.

Bit 9 = Left shift key status

When this bit equals one, the left shift key is pressed. When it is zero, the key is released.

Bit 10 = Right shift key status

When this bit equals one, the right shift key is pressed. When it is zero, the key is released.

Bit 11 = Left alt key status

When this bit equals one, the left alt key is pressed. When it is zero, the key is released.

Bit 12 = Right alt key status

When this bit equals one, the right alt key is pressed. When it is zero, the key is released.

Bits 13-15 = Reserved.

ID = 17, 57 Locator Report

Interrupt IDs 17 and 57 return status for the locator relative (mouse) and absolute (tablet) devices, the lighted programmable function keys, and dials. The format of the interrupt for each device type is similar. Bits 8-15 of old PSB word 5 indicate the device for which interrupt information is being returned.

Old PSB word 8 provides two granularities of time stamps. The three bytes of time stamp 1 contain the time since IPL of the machine in seconds. The time stamp 2 byte contains the current count of the VRM 60 Hz. counter. This extension provides a finer granularity of time-stamping facilities.

The formats of the status field and of old PSB word 7 vary by device, but provide information on device movement or the origin of device input. The format of interrupts 17 and 57 is shown in the following figures for each of the supported devices.

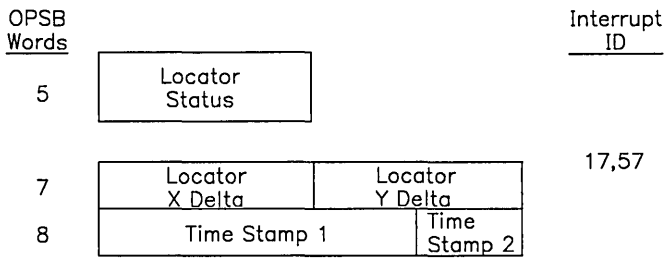


Figure 3-20. Locator Relative Report

When HOSTPC mode is set and a mouse device is configured, this interrupt returns information on mouse movement and the status of the buttons on the mouse.

The fields in the preceding figure are defined as follows:

- Locator status

  For a mouse (relative) device, the second byte of OPSB word 5 is set to 0. In addition, the value of bits 0-2 of OPSB word 5 indicate the mouse button being pressed. These bits are defined as follows:

  - Bit 0 = 1 : mouse button 1 pressed
  - Bit 1 = 1 : mouse button 2 pressed
  - Bit 2 = 1 : mouse button 3 pressed

  If locator buttons 1 and 2 are pressed at the same time, the command virtual terminal (if defined) will become activated and a locator-relative report with status showing both buttons pressed may be received by the virtual machine.

- Locator X delta

  The X delta refers to accumulated horizontal locator movement since the last locator movement. The unit of measurement is .25 mm.

- Locator Y delta

  The Y delta refers to accumulated vertical locator movement since the last locator movement. The unit of measurement is .25 mm.

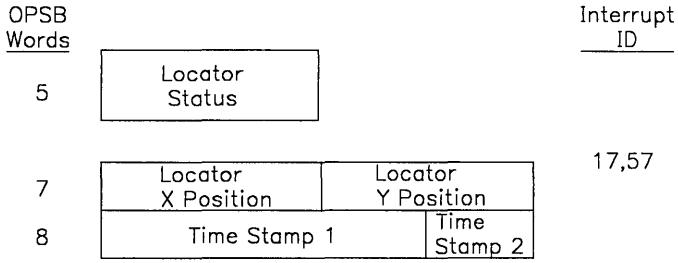


Figure 3-21. Locator Absolute Report

When HOSTPC mode is set and a tablet device is configured, this interrupt returns information on tablet movement and the status of the buttons (if a cursor tablet is configured).

The fields in the preceding figure are defined as follows:

- Locator status

  For a tablet (absolute) device, the second byte of OPSB word 5 is set to 1. Bit 5 indicates whether the tablet stylus or cursor is resting on the tablet (required to report input). For a cursor tablet only, the value of bits 0-4 of OPSB word 5 indicate the tablet buttons being pressed. For example, if bits 0-4 have a binary value of 00011, tablet button 3 is being pressed.

- Locator X Position

  The X position refers to the location of stylus or cursor on the horizontal axis.

- Locator Y Position

  The Y position refers to the location of stylus or cursor on the vertical axis.

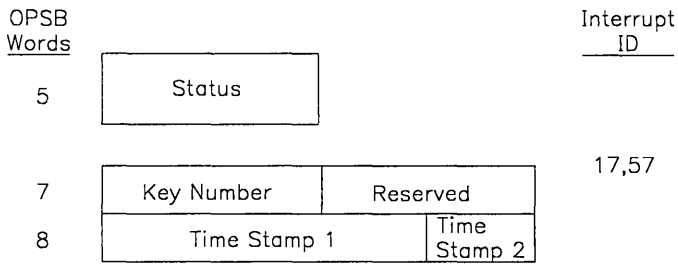


Figure 3-22. Lighted Programmable Function Key Report

When HOSTLPFK mode is set and the LPF key device is configured, this interrupt returns information on key number and status.

The fields in the preceding figure are defined as follows:

- Status

  For the LPF key device, the second byte of OPSB word 5 is set to 2.

- Key Number

  The value of this halfword indicates which of the LPF keys is reporting data.

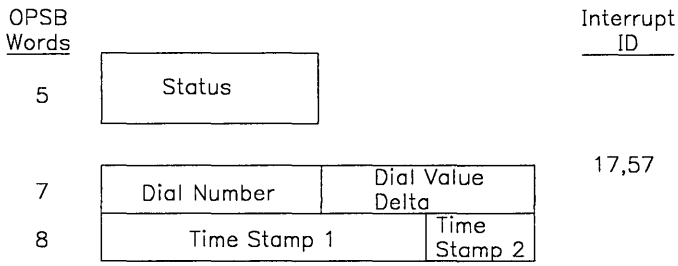


Figure 3-23. Dials Report

When HOSTDIALS mode is set and the dials device is configured, this interrupt returns information on dial number and status.

The fields in the preceding figure are defined as follows:

- Status

  For the dials, the second byte of OPSB word 5 is set to 3.

- Dial Number

  The value of this halfword indicates which of the dials is reporting data.

- Dial Value Delta

  This halfword indicates the direction and magnitude of movement for this dial since the last report. See "Set Dials" on page 3-62.

For the following unsolicited interrupts, the interrupt ID alone determines the required action. No other pertinent information is sent in the unsolicited interrupt control block.

ID = 20, 60 Screen grant

The VTMP returns this interrupt to advise the application (in monitored display screen mode) that the operating system will allocate the display screen to the application.

ID = 21, 61 Screen Release

The VTMP interrupts an application (in monitored display screen mode) to request that it relinquish control of the display screen so the operating system can allocate the screen to another application.

The VTMP starts a timer after it issues this unsolicited interrupt. If the application does not comply with the request before the time interval expires, the VTMP removes the application with another unsolicited interrupt. (See interrupt ID 22.)

ID = 22, 62 Remove application

This interrupt removes the uncooperative application that failed to comply with a screen release interrupt in the time allowed by the VTMP.

This interrupt is also sent if the virtual memory segment containing the application-defined monitored mode ring buffer is destroyed by way of a **Destroy Segment SVC.**

ID = 23, 63 Acknowledge sound data

The virtual terminal sends this interrupt to the application if the application set the acknowledge flag in a sound data request.

ID = 24, 64 MOM input buffer status change

The VTMP returns this unsolicited interrupt to the application when the monitored mode input ring changes from empty to non-empty. See "Screen Request and Input Ring Addressability" on page 3-100 for more information on how the MOM input ring works.

Adapter-Generated Interrupt Data

Some adapters can return status information to MOM applications by way of a ring buffer. This status information is placed in the ring buffer with a VTA escape sequence (ESC [r). Note that this feature is not available to KSR mode.

The information that immediately follows the escape sequence includes a 1-byte queue ID, and 20 bytes of data. Note that the hardware returns 16-bit words and that the bit-numbering conventions are reversed. See *IBM RT PC Hardware Technical Reference* for details on the data returned for each adapter status entry.

For information on special coding requirements for display adapters that generate interrupts, see "Coding Concepts for Adapters that Generate Interrupts" on page 3-200.

Status	QID	Data
FIFO mode entered	0x01	0x03 in first data byte, rest reserved.
PHIGS traversal started	0x01	0x05 in first data byte, rest reserved.
FIFO pick mode set	0x01	0x07 in first data byte, rest reserved.
CGA mode entered	0x01	0x09 in first data byte, rest reserved.
Traversal stopped	0x01	0x0B in first data byte, rest reserved.
Single-step mode completed	0x01	0x0F in first data byte, rest reserved.
Echo cursor completed	0x01	0x11 in first data byte, rest reserved.
Defined pointer echo completed	0x01	0x13 in first data byte, rest reserved.
Remove cursor completed	0x01	0x15 in first data byte, rest reserved.
Clear frame buffer completed	0x01	0x17 in first data byte, rest reserved.
Load look-up table completed	0x01	0x21 in first data byte, rest reserved.
Set pick window size completed	0x01	0x27 in first data byte, rest reserved.
Reset FIFO pick mode completed	0x01	0x29 in first data byte, rest reserved.
Set blink mode completed	0x01	0x2D in first data byte, rest reserved.
Reset blink mode completed	0x01	0x2F in first data byte, rest reserved.
Initialization complete	0x02	0x01 in first data byte, rest reserved.
Traversal complete	0x03	No data, all 20 bytes reserved.
Pick occurred	0x04	Data words 1-5 set to reason extension words 1-10.

Status	QID	Data
Buffer error FIFO overflow Illegal graphic order Illegal request code Invalid page Stack error Traversal error	0x05	Data words 1-5 set to reason extension words 1-10.
PELPRO task completed	0x06	No data, all 20 bytes reserved.
PELPRO pick	0x07	Data words 1-5 set to reason extension words 1-10.
PELPRO vertical synch.	0x08	No data, all 20 bytes reserved.
FIFO half full	0x09	No data, all 20 bytes reserved.
FIFO half empty	0x0A	No data, all 20 bytes reserved.
Synchronize	0x0B	Data words 1-5 set to reason extension codes 1-10.

Acknowledge Interrupt Return Codes

The following values may be found in the 'Return Code' halfword of the various acknowledge interrupt PSB. Note that all return code values are in the base 10 numbering system.

6516 = Unsuccessful, invalid virtual address detected by SVC handler
6517 = Unsuccessful, VTMP received an invalid queue element
6521 = Unsuccessful, invalid length specified in VTD block
6522 = Unsuccessful, invalid major type
6523 = Unsuccessful, invalid minor data
6524 = Unsuccessful, invalid minor type
6526 = Unsuccessful, invalid data type
6527 = Unsuccessful, VTD block exceeds 128K bytes
6528 = Unsuccessful, VTD block is less than the minimum length
6529 = Device not available
6531 = Unsuccessful, cannot remap a character set other than unique 1 or 2
6532 = Invalid locator type request
6533 = Unsuccessful, invalid font ID
6534 = Unsuccessful, fonts are not the same size
6535 = Unsuccessful, no font available for the designated display
6536 = Unsuccessful, virtual terminal driver returned an unexpected return code
6537 = Invalid graphics asynchronous device driver request
6538 = Specified device not configured
6539 = Specified device not selected
6544 = Unsuccessful, data received for an inactive mode
6545 = Unsuccessful, specified virtual terminal not active
6546 = Unsuccessful, invalid virtual terminal identifier
6548 = Unsuccessful, invalid coordinates specified in Query ASCII Codes command
6549 = Unsuccessful, invalid parameter detected in a control sequence
6550 = Unsuccessful, unsupported control sequence or code received
6555 = Unsuccessful, error from **_enque** routine
6556 = Unsuccessful, error from **_bind** routine
6557 = Unsuccessful, error from **_copy** routine
6561 = Unsuccessful, invalid structure selector
6562 = Unsuccessful, invalid echo map length
6563 = Unsuccessful, undefined function ID
6564 = Unsuccessful, cannot remap keys reserved for resource controller
6565 = Unsuccessful, invalid flags in the keyboard mapping structure
6566 = Unsuccessful, invalid key position
6570 = Unsuccessful, the previously defined monitored mode application input buffer no longer exists.
6580 = Unsuccessful, incomplete query command
6581 = Unsuccessful, query response buffer overflowed
6583 = Unsuccessful, invalid device identifier

Major Data Types

The output buffer from an SVC may contain different data structures that correspond to the major data types.

Virtual terminals accept three types of major data. They are:

- General
- Keyboard Send/Receive (KSR)
- Monitored Mode (MOM).

Each major data type presents data to the virtual terminal in the same structure. See Figure 3-24.

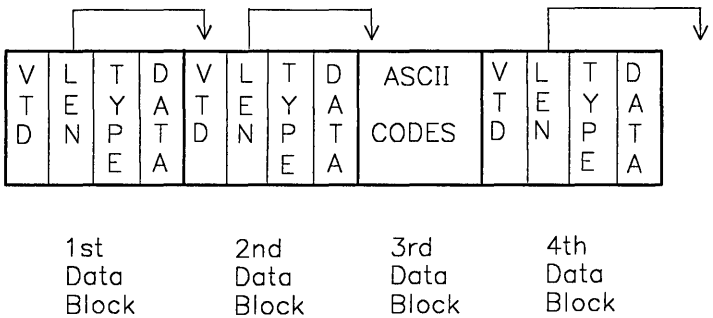


Figure 3-24. Virtual Terminal Data Stream Format

In the figure above, the 3rd data block is interpreted as type = KSR/ASCII.

Each structure is introduced by a virtual terminal data (VTD) control sequence.

The VTD control sequence is defined as the ASCII codes 'ESC', '[', and 'x' (0x1B5B78). Figure 3-25 illustrates the VTD.

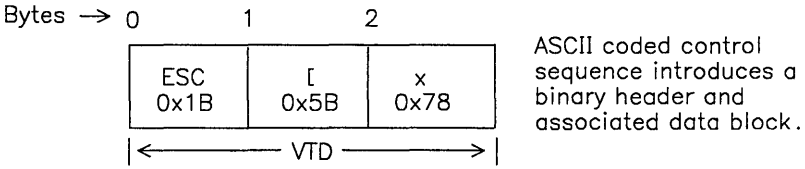


Figure 3-25. Virtual Terminal Data (VTD) Control Sequence

The display data structure, shown in Figure 3-26 on page 3-57, follows the VTD control sequence. Four bytes of length information follow the VTD, and two bytes of type information follow the length field. Note that the length of any VTD (bytes 0-4) is limited to 128K bytes.

The first type byte specifies a major data type. Major data types are KSR, MOM, and general (for either KSR or MOM). The second type byte specifies a minor type within the major type. The length bytes are interpreted as 32-bit integers which specify the total number of bytes in the header (not including the VTD control sequence).

If a virtual terminal receives data of a conflicting type, such as MOM data following KSR data, the VTMP does not switch modes, but returns an error code to the virtual machine.

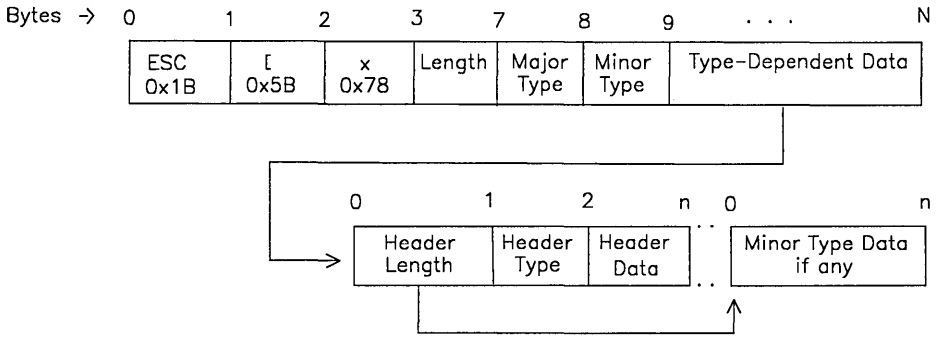


Figure 3-26. Display Data Structure

The fields in the preceding figure are defined as follows:

Length — Indicates the total number of bytes in the header and associated data (not including the VTD control sequence). Therefore, total length in bytes equals N - 2.

Major type — Indicates the major type of data presented to the virtual terminal. Possible values for this field are:

0x00 Reserved
0x01 General
0x02 KSR
0x05 Monitored

For more information on major data types, see "General Major Data Type" on page 3-58, "KSR Major Data Type" on page 3-69, and "MOM Major Data Type" on page 3-98.

Minor type — Indicates a minor type within the major type. Minor type values are further defined as follows:

0-63 Data passed for output
64-127 Data passed to specify query requests
128-191 Reserved
192-255 Query response (acknowledgement) data.

Type-dependent data — Indicates additional command-dependent data.

General Major Data Type

General data types are those that are not specific to a single (KSR or MOM) data type.

A major type value of 0x01 identifies general major data. The minor data types associated with general major data are summarized in the following table and described in subsequent sections.

Value	Minor Type
0x06	Keyboard light-emitting diode (LED) settings
0x07	Locator X and Y thresholds (mouse)
0x08	Set non-input zone (tablet)
0x09	Sound data
0x0A	Cancel sound data
0x0C	Set LPF keys
0x0D	Set dials
0x16	Change physical display
0x40	Query physical device IDs
0x41	Query physical device
0x42	Query locator
0x43	Query LPF keys
0x44	Query dials
0x45	Query terminal
0x46	Query DMA
0xC0	Acknowledge query physical device IDs
0xC1	Acknowledge query physical display
0xC2	Acknowledge query locator
0xC3	Acknowledge query LPF keys
0xC4	Acknowledge query dials
0xC5	Acknowledge query terminal
0xC6	Acknowledge query DMA.

Note: All reserved fields must be set equal to zero.

Set Light-Emitting Diodes (LEDs)

You can change the settings of the LEDs provided with the RT PC keyboard.

This field appears at the start of minor type data and is defined as follows:

Bytes Value

0-1 Set or reset LEDs

The following bits indicate whether to use the LED defaults or to set new LED values.

Bit 0

When bit 0 equals 0, the LED state remains unchanged. When bit 0 equals 1, you specify in bit 8 whether to set or reset the leftmost LED. The leftmost LED on the keyboard is for the **Num Lock** key.

Bit 8 = 0 — reset the leftmost LED
Bit 8 = 1 — set leftmost LED

Bit 1

When bit 1 equals 0, the LED state remains unchanged. When bit 1 equals 1, you specify in bit 9 whether to set or reset the middle LED. The middle LED on the keyboard is labeled **Caps Lock**.

Bit 9 = 0 — reset the middle LED
Bit 9 = 1 — set the middle LED

Bit 2

When bit 2 equals 0, the LED state remains unchanged. When bit 2 equals 1, you specify in bit 10 whether to set or reset the rightmost LED. The rightmost LED on the keyboard is labeled **Scroll Lock**.

Bit 10 = 0 — reset the rightmost LED
Bit 10 = 1 — set the rightmost LED

Locator Thresholds

The locator device driver receives notice of horizontal and vertical mouse movement. The delta of these movement events are monitored by the driver, until the accumulated events exceed either the horizontal or vertical thresholds, or both. The locator device driver accumulates measurements at consecutive samplings. When a threshold is exceeded, the driver enqueues the information to the VTMP.

The accumulated measurements can also be returned to the VTMP, even if they do not exceed a threshold, if the status of the mouse buttons changes.

Either way, the VTMP passes the information back to the application. The VTMP provides neither echoing nor positional management functions for the locator.

Each opened virtual terminal has its own threshold values. When a virtual terminal is opened, the threshold values default to 2.75 mm of horizontal and 5.5 mm of vertical physical locator movement.

If you set the thresholds to zero, each event report is returned to the VTMP at the sampling rate supported by the locator device driver.

You can disable locator input completely by using the HOSTPC bit in the protocol mode definition or by setting both the thresholds to the maximum values.

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0-1	Horizontal threshold

You can specify the horizontal threshold in values from 0 to 32K locator units. A locator unit on the IBM-supplied locator equals .25 mm.

2-3 Vertical threshold

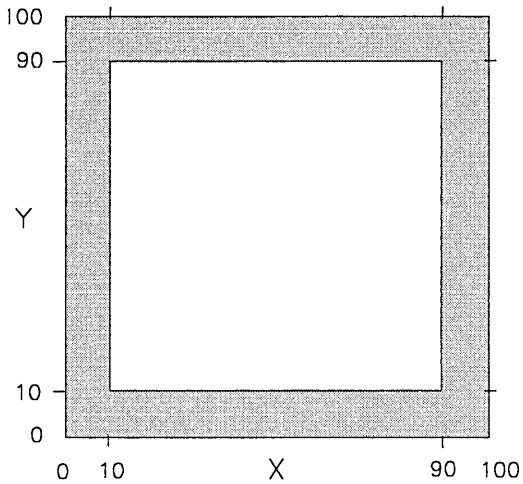
You can specify the vertical threshold in values from 0 to 32K locator units. A locator unit on the IBM-supplied locator equals .25 mm.

Set No-Input Zone (Tablet)

A no-input zone is an area on a tablet device from which no reports can be input to the VTMP. No-input zones are maintained for each terminal (not a global attribute) to reduce the area of the physical tablet from which reports can be made.

The locator device driver supports both a horizontal (X delta) and a vertical (Y delta) no-input zone. In addition, the device driver determines the units of granularity for both the X and Y values.

When a virtual terminal is opened, the no-input values in the VTMP default to zero, meaning the entire tablet surface will report input. The following figure shows an example of a tablet with a no-input zone defined. The shaded area represents the no-input zone.



This field appears at the start of minor type data and is defined as follows:

Bytes Value

0-1 Horizontal no-input zone

This field specifies the horizontal no-input zone in values from 0 to 32K units. The locator device driver defines the value of each unit.

2-3 Vertical no-input zone

This field specifies the vertical no-input zone in values from 0 to 32K units. The locator device driver defines the value of each unit.

Sound

This command sends output to the speaker. The mode byte determines whether to execute sound commands for the active virtual terminal and whether to interrupt the application after the sound command executes.

No range check is made for the frequency or duration values. However, frequency values below 23 hz and above 12K hz will be timed correctly but emit no sound from the speaker.

This field appears at the start of minor type data and is defined as follows:

Bytes Value

0 Mode

Bits 0 and 7 determine the acknowledge data and mode, respectively.

Bit 0 = 0 — Do not acknowledge when sound command executes.
Bit 0 = 1 — Acknowledge when sound command executes or is discarded.

Bit 7 = 0 — Play sound only if terminal is active.
Bit 7 = 1 — Play all sound to this terminal.

1 Reserved

2-3 Duration in 1/128 seconds

4-5 Frequency in hertz.

Cancel Sound

The cancel sound command removes all enqueued sound requests from the speaker device driver that have been marked 'play sound only when active'. This leaves only the commands that request 'play all sound to this terminal' in the active terminal's queue. An inactive terminal ignores this command. The active terminal sends the cancel command to the speaker driver in a high priority queue element, followed by an enable sound request in a normal priority queue element. When the speaker driver receives the cancel sound command, it purges its queue of the elements that do not have the execute sound flag set (bit 7 of sound command byte 0 equals 1).

The speaker driver's queue is flushed whenever a virtual terminal transition occurs by sending this cancel sound/enable sound queue element pair. Regardless of whether the sound request is executed or purged, the virtual terminal receives an acknowledgement if the acknowledge flag is set (bit 0 of sound command byte 0 equals 1).

A VTD header, length field, major type field and minor type field are required to issue this command. No other parameters are necessary.

Set LPF Keys

You can set the LPF key lights on or off. The default setting is off.

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0-3	Reserved
4-7	LPF key mask Bits 0-31 represent 32 possible LPF keys. When any of bits 0-31 are set (equal to one), the corresponding bit in bytes 8-11 determines whether the key is to be lit.
8-11	LPF key flags Bits 0-31 represent 32 possible LPF keys. If any of bits 0-31 are set (equal to one), the key is to be lit.
12-27	Reserved.

Set Dials

You can change the granularity of the dials. Dial granularity means the number of events that occur for a full (360 degree) revolution of the dial. The dial granularity values represent powers of 2 (2^n), where $2 \leq n \leq 8$. The default granularity is 4.

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0-3	Reserved
4-5	Dial mask Bits 0-16 represent 16 possible dials. When any of bits 0-16 are set (equal to one), the corresponding byte from bytes 8-23 indicates the new granularity.
6-7	Reserved
8-23	Dial granularity values Bytes 8-23 correspond to bits 0-16 of bytes 4-5. When any of these bits are set, the corresponding byte contains the value to use as the number of events per full revolution of the dial.
24-27	Reserved.

Change Physical Display

You can use this command to change the physical display to which a virtual terminal is logically attached.

This field appears at the start of minor type data and is defined as follows:

Bytes Value

0-1 Flags

Bits 0-1 and 3-15 of these bytes are reserved. Bit 2 specifies the default or a value you specify for the physical display.

Bit 2 = 0 — Use the current physical screen.
Bit 2 = 1 — Use the identifier specified in bytes 10-13 for the physical display.

2-9 Reserved

10-13 Physical display device identifier.

14-23 Reserved

Query Physical Device IDs

You specify this command with a VTD header, length field, major type field and minor type field. No other parameters are needed.

Note that this command can be issued only by way of the **VT Query SVC**.

Acknowledge Query Device IDs

This command tells you how many devices are returned, then gives you the device ID and display class for each returned device.

This field appears at the start of minor type data and is defined as follows:

Bytes Value

0-1 The number of devices for which data is returned

Bytes 2 through 9 are repeated for each physical device returned (the value of bytes 0-1).

2-5 Physical device ID

These four bytes uniquely identify the display adapters presently configured. The first device ID is the active display device ID, unless the Change Physical Display command has changed the active display ID. Possible values for this field are:

0x0401mmnn IBM PC Monochrome Display Adapter
0x0402mmnn IBM APA8 Monochrome Display Adapter
0x0403mmnn IBM EGA Monochrome Display Adapter
0x0404mmnn IBM EGA Color Display Adapter
0x0405mmnn IBM APA16 Monochrome Display Adapter

0x0406mmnn IBM APA8 Color Display Adapter
0x0408mmnn IBM 5081 Display Adapter

In the preceding list, the 'mm' value indicates if the adapter is totally functional. When 'mm' equals 0x00, the adapter is totally functional. Any value other than 0x00 in this field indicates the adapter is less than totally functional (perhaps not working at all) but is present on the machine.

The 'nn' value in the preceding list can be in the range 0x01 to 0x04. This value is used to differentiate between multiple instances of the same adapter type on the machine.

6-9 Display class (0x44)

The returned IDs can be used with the **Query Physical Device** and **Change Physical Display** commands.

Query Physical Device

You can determine information concerning display or locator devices with this command.

This field appears at the start of minor type data and is defined as follows:

Bytes Value

0-3 Physical device ID

You can query the characteristics of the device to which you are currently attached by specifying 0 here.

4-7 Reserved (set equal to zero)

Note that this command can be issued only by way of the **VT Query SVC**.

Query Locator

You specify this request with a VTD header, length field, and major and minor type field. Information about the configured device (mouse or tablet, if any) will be returned.

Query LPF Keys

You specify this request with a VTD header, length field, and major and minor type field. Note that LPF keys must be enabled before they can be successfully queried.

Query Dials

You specify this request with a VTD header, length field, and major and minor type field. Note that dials must be enabled before they can be successfully queried.

Query Terminal

This query is used to determine information concerning the terminal, keyboard, font, color, and presentation size. The presentation space size may vary from font to font. You specify this request with a VTD header, length field, and major and minor type field.

Query DMA

This query is used to determine the starting address and length of the application's DMA area. You specify this request with a VTD header, length field, and major and minor type field.

Acknowledge Query Physical Device

When you issue a Query Physical Device, you get the results of the query by way of an Acknowledge Query Physical Device. This command returns the VTD header, major and minor data types, 2 bytes of minor data header, followed immediately by a Locator Device Acknowledge, Display Device Acknowledge, Display Device Font Acknowledge, and Display Device Color Acknowledge. The format of this acknowledge structure is defined as follows:

Locator Device Acknowledge

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0-3	Scale factor (millimeters per 100 counts)
4	Device type Bit 0 = 0 - relative device Bit 0 = 1 - absolute device
5-7	Reserved (set equal to 0)

Display Device Acknowledge

This structure immediately follows the locator device acknowledge structure and is defined as follows:

Bytes	Value
8-11	Display device attributes Bit 0 = 0 — character display Bit 0 = 1 — all-points-addressable (APA) display Bit 1 = 0 — no blink function Bit 1 = 1 — blink function allowed Bit 23 = 0 — no color capability Bit 23 = 1 — color allowed

Bit 24 = 0 — cannot change display adapter's color palette
Bit 24 = 1 — can change display adapter's color palette

All other bits are reserved and set equal to zero.

12-15 Displayable width of physical screen

This value is expressed in picture elements (pels) for all displays.

16-19 Displayable height of physical screen

This value is expressed in pels for all displays.

20-23 Displayable width (in millimeters)

24-27 Displayable height (in millimeters)

28-31 Bits per pel (1, 2 or 4)

32-35 The ID of the physical device being queried.

Display Device Font Acknowledge

This structure immediately follows the display device acknowledge structure:

Bytes Value

36-39 Number of fonts available to this display

The following six fields appear for each available font.

40-43 Physical font ID

44-47 Physical font style

This field is always zero.

48-51 Physical font attribute

This field may have the following values:

0 = plain
1 = bold
2 = italic.

52-55 Physical font width (the width of a character box in pels)

56-59 Physical font height (the height of a character box in pels)

60-nn Start of next font (if applicable).

Display Device Color Acknowledge

This structure immediately follows the display device font acknowledge structure and is an offset from the end of the final font acknowledge:

Offset	Value
0-3	Total number of colors possible
4-7	Number of colors that can be active at any one time
8-11	Number of foreground color options
12-15	Number of background color options
16-19	Default setting for the color palette For color devices, this field represents the color associated with the KSR-defined color for SGR entries. This field is repeated for each of the currently active colors.
20-nn	Next active color value (if applicable).

Acknowledge Query Locator

This command tells you whether a mouse or tablet is defined and provides threshold or no-input zone data as appropriate. If a tablet is defined, it indicates whether the tablet sensor is a stylus or cursor.

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0-3	Locator resolution
4	Locator device Bit 0 = 0 - mouse Bit 0 = 1 - tablet Bits 1,2 - indicate sensor type for tablet only = 00 - unknown = 01 - stylus = 11 - cursor Bits 4-7 are reserved.
5-7	Reserved
8-9	Maximum horizontal count
10-11	Maximum vertical count
12-13	Horizontal no-input zone (tablet) or threshold (mouse)
14-15	Vertical no-input zone (tablet) or threshold (mouse)
16-23	Reserved

Acknowledge Query LPF Keys

This command tells you how many keys are on the LPF keyboard and also whether the light for each key is on or off.

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0-3	Number of LPF keys on the device
4-7	Reserved
8-11	LPF key flags Bits 0-31 comprise a mask for 32 possible LPF keys. When any of the bits are set (equal to one), the key is lighted.
12-27	Reserved

Acknowledge Query Dials

This command tells you how many dials are on the device and also the granularity of each dial.

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0-3	Number of dials on the device
4-7	Reserved
8-23	Granularity values Bytes 8-23 indicate the granularity value for up to 16 dials.
24-27	Reserved

Acknowledge Query Terminal

This command tells you the number of rows and columns that logically exist for the defined font on the configured display.

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0-3	Physical display device ID
4-7	Number of rows
8-11	Number of columns
12-15	Number of colors

16-19 Number of fonts

20-23 Physical keyboard ID

Possible values for this field include:

- 0 = 101-key keyboard
- 1 = 102-key keyboard
- 2 = 106-key keyboard

24-31 Reserved

Acknowledge Query DMA

This command tells you the starting address and the length of the application's DMA space.

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0-3	Starting address of DMA space
4-7	Length of DMA space

KSR Major Data Type

KSR data is recognized by a major type field of value 0x02, or by the absence of a VTD control following a data block. The minor data types associated with KSR major data are summarized in the following table and described in subsequent sections.

Value	Minor Type
0x00	ASCII codes (assumed in absence of VTD and header)
0x01	KSR protocol mode definition
0x02	Character set definition
0x05	Set KSR Color Palette
0x06	Change fonts
0x08	Text Cursor Representation
0x40	Query ASCII code and attribute
0x80	ASCII interrupt codes
0x84	Key position, status, scan code interrupt
0x91	Locator relative report interrupt.

ASCII Codes

ASCII codes represent displayable graphics or single-byte control functions. Certain sequences of codes represent ANSI-defined multi-byte control functions. Controls tell the KSR virtual terminal how to manage its presentation space. Presentation space is the maximum number of lines and characters that will fit on the selected physical display screen using the selected font. Graphic codes are converted to display symbols which are placed into the presentation space.

Character Codes

Graphic character codes passed in KSR data have the 8-bit RTASCII format (0x20-0xFF). These codes are translated into a 10-bit display font code according to the active G0 character set. A Shift Out control code causes output to be translated according to the G1 character set. The Shift In and Shift Out codes are in effect for translating succeeding data streams until another code is received. The single-shift codes (SS1, SS2, SS3, and SS4) affect only the next 8-bit code to be translated according to the code page indicated by the control.

Translated input from code page P0 is reported as an 8-bit code. Codes from other code pages are preceded by one of the single-shift control codes. These codes are single shift 1 (0x1F), single shift 2 (0x1E), single shift 3 (0x1D), and single shift 4 (0x1C). The single shift codes determine the code page to use for interpreting the code. With the exception of the single-byte controls (0x00 - 0x1F), the codes that immediately follow a single-shift control have the high-order bit set. For example, the code 0x20 from code page P0 is reported as 0x20, but 0x20 reported from code page P1 is reported as 0x1FA0. Thus, a code that comes from the lower half (0x20-0x7F) of either code page P1 or P2 is reported with its high-order bit artificially set.

The application that examines input has to strip the high-order bit to determine the code point/graphic symbol relationship if the application is scanning for specific graphics. If the application is not scanning for specific graphics, the input data stream reported to the virtual machine may be redirected as output back to the VTMP without alteration (bit stripping) in order to receive an exact copy of the display screen.

Input codes which are flagged as non-spacing characters are held for combinative processing according to the rules specified in the following section.

Non-Escaping Codes

Escapement refers to the logical forward movement of the cursor. Some keyboard layouts (for countries other than the United States) have certain code points that represent non-escaping characters. These characters are members of a finite set of diacritic symbols that the VTMP supports. These diacritics are defined as follows:

Diacritic	Code Page - Hex Value
Acute	P0 - 0x27
	P0 - 0xEF
Grave	P0 - 0x60

Circumflex	P0 - 0x5E
Umlaut	P0 - 0xF9
Tilde	P0 - 0x7E
Caron	P1 - 0x73
Breve	P1 - 0x9D
Double acute	P1 - 0x9E
Overcircle	P1 - 0x7D
Overdot	P1 - 0x85
Macron	P1 - 0xA3
Cedilla	P0 - 0xF7
Ogonek	P1 - 0x87

When any of these characters is defined to be non-escaping (does not cause a logical movement of the cursor) and is entered from the keyboard, the character is not returned immediately in an interrupt, but is held until the next keystroke is entered. In addition, if the character is normally echoed to the display, it will not immediately appear on the display. The VTMP tries to combine these characters with the next keystroke, which should be an alphabetic character. The VTMP will report and display only the combined character. Echoing to the display occurs only if the combined character has its flag set for echo.

In addition to the finite list of non-escaping characters, the following finite sets of valid combinations are defined:

Acute Accented Characters

´	Acute Accent	P0-0xEF
'	Apostrophe (Acute)	P0-0x27
é	e Acute Small	P0-0x82
É	e Acute Capital	P0-0x90
á	a Acute Small	P0-0xA0
í	i Acute Small	P0-0xA1
ó	o Acute Small	P0-0xA2
ú	u Acute Small	P0-0xA3
Á	a Acute Capital	P0-0xB5
Í	i Acute Capital	P0-0xD6
ý	y Acute Small	P0-0xEC
Y´	y Acute Capital	P0-0xED

Acute Accented Characters

Ó	o Acute Capital	P0-0xE0
Ú	u Acute Capital	P0-0xE9
ć	c Acute Small	P1-0x6A
ĺ	l Acute Small	P1-0x6E
Ć	c Acute Capital	P1-0x72
Ĺ	l Acute Capital	P1-0x77
ś	s Acute Small	P1-0x7C
ń	n Acute Small	P1-0x7F
Ś	s Acute Capital	P1-0x84
ź	z Acute Small	P1-0x8A
Ź	z Acute Capital	P1-0x8C
ŕ	r Acute Small	P1-0x91
Ŕ	r Acute Capital	P1-0x95
ǵ	g Acute Small	P1-0xB1

Grave Accented Characters

`	Grave Accent	P0-0x60
à	a Grave Small	P0-0x85
è	e Grave Small	P0-0x8A
ì	i Grave Small	P0-0x8D
ò	o Grave Small	P0-0x95
ù	u Grave Small	P0-0x97
À	a Grave Capital	P0-0xB7
È	e Grave Capital	P0-0xD4
Ì	i Grave Capital	P0-0xDE
Ò	o Grave Capital	P0-0xE3
Ù	u Grave Capital	P0-0xEB

Circumflex Accented Characters

^	Circumflex Accent	P0-0x5E
â	a Circumflex Small	P0-0x83
ê	e Circumflex Small	P0-0x88
î	i Circumflex Small	P0-0x8C
ô	o Circumflex Small	P0-0x93
û	u Circumflex Small	P0-0x96
Â	a Circumflex Capital	P0-0xB6
Ê	e Circumflex Capital	P0-0xD2
Î	i Circumflex Capital	P0-0xD7
Ô	o Circumflex Capital	P0-0xE2
Û	u Circumflex Capital	P0-0xEA
ĉ	c Circumflex Small	P1-0xA8
Ĉ	c Circumflex Capital	P1-0xA9
ĝ	g Circumflex Small	P1-0xB2
Ĝ	g Circumflex Capital	P1-0xB3
ĥ	h Circumflex Small	P1-0xB7
Ĥ	h Circumflex Capital	P1-0xB8
ĵ	j Circumflex Small	P1-0xC3
Ĵ	j Circumflex Capital	P1-0xC4
ŝ	s Circumflex Small	P1-0xD6
Ŝ	s Circumflex Capital	P1-0xD7
ŵ	w Circumflex Small	P1-0xE2
Ŵ	w Circumflex Capital	P1-0xE3
ŷ	y Circumflex Small	P1-0xE4
Ŷ	y Circumflex Capital	P1-0xE5

Umlaut Accented Characters

¨	Umlaut Accent	P0-0xF9
Ü	u Umlaut Capital	P0-0x9A
ü	u Umlaut Small	P0-0x81
ä	a Umlaut Small	P0-0x84
ë	e Umlaut Small	P0-0x89
ï	i Umlaut Small	P0-0x8B
Ä	a Umlaut Capital	P0-0x8E
ö	o Umlaut Small	P1-0x94
ÿ	y Umlaut Small	P1-0x98
Ö	O Umlaut Capital	P0-0x99
Ü	u Umlaut Capital	P0-0x9A
Ë	e Umlaut Capital	P0-0xD3
Ï	i Umlaut Capital	P0-0xD8
Ÿ	y Umlaut Capital	P1-0xE6

Tilde Accented Characters

~	Tilde Accent	P0-0x7E
ñ	n Tilde Small	P0-0xA4
Ñ	n Tilde Capital	P0-0xA5
ã	a Tilde Small	P0-0xC6
Ã	a Tilde Capital	P0-0xC7
õ	o Tilde Small	P0-0xE4
Õ	o Tilde Capital	P0-0xE5
ĩ	i Tilde Small	P1-0xBB
Ĩ	i Tilde Capital	P1-0xBC
ũ	u Tilde Small	P1-0xDA
Ũ	u Tilde Capital	P1-0xDB

Caron Accented Characters

ˇ	Caron Accent	P1-0x73
ě	e Caron Small	P1-0x68
ň	n Caron Small	P1-0x69
ď	d Caron Small	P1-0x6D
Ě	e Caron Capital	P1-0x70
Č	c Caron Capital	P1-0x71
Ď	d Caron Capital	P1-0x76
Ľ	l Caron Small	P1-0x78
ň	n Caron Small	P1-0x79
ř	r Caron Small	P1-0x7B
š	s Caron Small	P1-0x80
Ľ	l Caron Capital	P1-0x81
Ň	n Caron Capital	P1-0x82
Ř	r Caron Capital	P1-0x83
ž	z Caron Small	P1-0x89
Ž	z Caron Capital	P1-0x8B
Š	s Caron Capital	P1-0x8F
ť	t Caron Small	P1-0x90
Ť	t Caron Capital	P1-0x94

Breve Accented Characters

˘	Breve Accent	P1-0x9D
ă	a Breve Small	P1-0x98
ğ	g Breve Small	P1-0x99
Ă	a Breve Capital	P1-0x9B
Ğ	g Breve Capital	P1-0x9C
ŭ	u Breve Small	P1-0xDC
Ŭ	u Breve Capital	P1-0xDD

Double Acute Accented Characters

˝	Double Acute Accent	P1-0x9E
ő	o Double Acute Small	P1-0x92
ű	u Double Acute Small	P1-0x93
Ő	o Double Acute Capital	P1-0x96
Ű	Double Acute Capital	P1-0x97

Overcircle Accented Characters

°	Overcircle Accent	P1-0x7D
å	a Overcircle Small	P0-0x86
Å	a Overcircle Capital	P0-0x8F
ů	u Overcircle Small	P1-0x6C
Ů	u Overcircle Capital	P1-0x75

Overdot Accented Characters

˙	Overdot Accent	P1-0x85
ż	z Overdot Small	P1-0x86
Ż	z Overdot Capital	P1-0x88
İ	i Overdot Capital	P1-0x9A
ċ	c Overdot Small	P1-0xAB
Ċ	c Overdot Capital	P1-0xAC
ė	e Overdot Small	P1-0xAD
Ė	e Overdot Capital	P1-0xAE
ġ	g Overdot Small	P1-0xB4
Ġ	g Overdot Capital	P1-0xB5

Macron Accented Characters

¯	Macron Accent	P1-0xA3
ā	a Macron Small	P1-0xA6
Ā	a Macron Capital	P1-0xA7
ē	e Macron Small	P1-0xAF
Ē	e Macron Capital	P1-0xB0
ī	i Macron Small	P1-0xBD
Ī	i Macron Capital	P1-0xBE
ō	o Macron Small	P1-0xD0
Ō	o Macron Capital	P1-0xD1
ū	u Macron Small	P1-0xDE
Ū	u Macron Capital	P1-0xDF

Cedilla Accented Characters

¸	Cedilla Accent	P0-0xF7
Ç	c Cedilla Capital	P0-0x80
ç	c Cedilla Small	P0-0x87
ş	s Cedilla Small	P1-0x9F
Ş	s Cedilla Capital	P1-0xA2
ţ	t Cedilla Small	P1-0xA4
Ţ	t Cedilla Capital	P1-0xA5
Ģ	g Cedilla Capital	P1-0xB6
ķ	k Cedilla Small	P1-0xC5
Ķ	k Cedilla Capital	P1-0xC6
ļ	l Cedilla Small	P1-0xC8
Ļ	l Cedilla Capital	P1-0xC9
ņ	n Cedilla Small	P1-0xCC
Ņ	n Cedilla Capital	P1-0xCD
ŗ	r Cedilla Small	P1-0xD4
Ŗ	r Cedilla Capital	P1-0xD5

Ogonek Accented Characters

˛	Ogonek Accent	P1-0x87
ą	a Ogonek Small	P1-0x67
ę	e Ogonek Small	P1-0x6B
Ą	a Ogonek Capital	P1-0x6F
Ę	e Ogonek Capital	P1-0x74
į	i Ogonek Small	P1-0xBF
Į	i Ogonek Capital	P1-0xC0
ų	u Ogonek Small	P1-0xE0
Ų	u Ogonek Capital	P1-0xE1

For more information on diacritics and other international keyboard data, see *IBM RT PC Keyboard Description and Character Reference.*

If a valid non-spacing character is followed by a keystroke that does not produce one of the valid combinations defined above, both the code and the following keystroke are returned, as is, in an unsolicited interrupt. Note that a blank space also produces a valid combination, yielding only the diacritic. In this case, only the diacritic is returned in the interrupt. Therefore, you should assume that all echoed characters that have been reported have caused escapement.

The non-escaping nature of a diacritic is not inherent in the code point, but is established as a characteristic of the code point in the default language software keyboards. If you remap a keyboard with the **VT Set Structure SVC**, you can make diacritic code points escaping or non-escaping, regardless of the attached physical keyboard.

Note that escaping character processing is an input-only function. No such concept exists for output processing.

The VTMP recognizes five predefined character set-to-display symbol mappings for 8-bit RTASCII. Four of the character sets contain 224 code points each, and one character set contains only 97 code points. These five code sets map into the display symbol map of 1024 symbols, and are described briefly below.

1. Display Symbols 32-255

   This code page represents the 7-bit ASCII characters, plus many international characters and some PCASCII line drawing characters.

2. Display Symbols 256-479

   These codes specify the PCASCII graphical representation for the range 0x01 through 0x1F, the teletext characters, and the EBCDIC characters from EBCDIC character sets 256, 257, 258, 330, 435 and 500 which are not already represented in the first character set.

3. Display Symbols 480-703

   These codes represent the portion of code page 259 that is not already represented in the display symbols 32-479, plus the set of reserved code points for possible support of additional code pages.

4. Display Symbols 704-927

   Reserved for user-defined symbols.

5. Display Symbols 928-1023

   Reserved for user-defined symbols.

In addition to the predefined character sets, the VTMP maintains two unique character set mappings that you can create from any of the predefined symbols. These sets are called 'Unique One' and 'Unique Two' and are identified by the leading characters ';' and ':', respectively. These characters have the following meanings:

; Unique One
: Unique Two

Until they are first modified, both user-definable sets are identical to the P0 code page. Either set may be modified by designating its ANSI final character (';' or ':'). Any or all of the positions in a set may be replaced with any of the 10-bit display symbols. The remap is defined to begin at position 00. Therefore, to remap a single position, you may either send the entire character map with only one position deviating from the current map, or you can send a map up to and including the one remapped character. All subsequent remappings to the same unique set will be applied over the existing map. The actual graphical representation of a display symbol is dependent on the hardware in use.

Character Code Processing

Graphic codes input from the keyboard or received by a KSR virtual terminal across the VMI are normally placed into the virtual terminal's data structure for viewing on a physical display device.

A KSR virtual terminal has a presentation space (PS) of a fixed number of columns per line, and a fixed number of lines. A symbol may be placed at any column on any line in the terminal's presentation space. Graphics from the KSR data stream are placed in the PS relative to the current cursor position. Keyboard input also relates to the cursor position.

Two modes for displaying graphics are replace and insert. In replace mode, a graphic character sent to a KSR terminal is placed above the cursor, writing over (replacing) whatever symbol was there. In insert mode, a graphic character sent to a KSR terminal is also placed above the cursor, but the symbol that had been above the cursor and all symbols on the same line are shifted right one column position on the line. Characters shifted out of the last column on the line are lost.

Another mode determines what happens to the cursor when characters fill a line, causing the cursor to reach the last column position on a line. This mode, automatic new line (AUTONL), determines if the cursor wraps around to the first column position of the next line or stays at the last column on the current line.

If AUTONL is set, the cursor moves to the first column position of the following line. If the cursor happens to be on the bottom line of the presentation space, the presentation space scrolls up one line. If AUTONL is reset, the cursor stays on the last column of the current line, and characters will continue to be displayed at that position.

Single Code ASCII Controls (ANSI Mode)

Listed here are the ASCII single code controls and their interpretation in KSR coded data. A line introducing each control gives its mnemonic, its code value, and its function.

- NUL 0x00 Null

  Has no terminal function.

- SOH 0x01 Start of Header

  Has no terminal function.

- STX 0x02 Start of Text

  Has no terminal function.

- ETX 0x03 End of Text

  Has no terminal function.

- EOT 0x04 End of Transmission

  Has no terminal function.

- ENQ 0x05 Enquiry

  Has no terminal function.

- ACK 0x06 Acknowledge

  Has no terminal function.

- BEL 0x07 Bell

  Causes an audible alarm to sound.

- BS 0x08 Backspace

  Moves the cursor position to the left one column on the same line. If the cursor is at the PS left boundary, the cursor position does not change.

- HT 0x09 Horizontal Tab

  Moves the cursor position forward to the next tab stop on the same line.

- LF 0x0A Line Feed

  If the LNM mode is reset, the line feed moves the cursor position down one line. If the LNM mode is set (default), the line feed is treated as a NEL and moves the cursor position to the first position of the next line. In either case, if the cursor is already on the last line of the PS, the PS lines scroll up one line. The top line of the PS disappears and a blank line is inserted as the new bottom line.

- VT 0x0B Vertical Tab

Moves the cursor position down to the next line that is defined as a vertical tab stop. Tabs stops are always set at the first and last lines of the PS. If the cursor was already on the last line of the PS and WRAP mode is not set, the cursor stays on the last line in the PS. If WRAP mode is set, the cursor moves to the top line in the PS. The column position does not change in any case.

- FF 0x0C Form Feed

  Treated as a line end; see NEL.

- CR 0x0D Carriage Return

  If the CNM mode is reset (default), the carriage return moves the cursor position to the first character of the cursored line. If the CNM mode is set, the carriage return is treated as an NEL and causes the cursor position to move to the first position of the next line. In this case, if the cursor is already on the last line of the PS, the PS lines scroll up one line. The top line of the PS disappears and a blank line is inserted as the new bottom line.

- SO 0x0E Shift Out

  Maps the subsequently received graphic codes to display symbols according to the active G1 character set.

- SI 0x0F Shift In

  Maps the subsequently received graphic codes to display symbols according to the active G0 character set.

- DLE 0x10 Data Link Escape

  Has no terminal function.

- DC1 0x11 Device Control 1

  Has no terminal function.

- DC2 0x12 Device Control 2

  Has no terminal function.
- DC3 0x13 Device Control 3

  Has no terminal function.
- DC4 0x14 Device Control 4

  Has no terminal function.
- NAK 0x15 Negative Acknowledgment

  Has no terminal function.
- SYN 0x16 Synchronous

  Has no terminal function.
- ETB 0x17 End of Block

  Has no terminal function.
- CAN 0x18 Cancel

  Has no terminal function.
- EM 0x19 End of Medium

  Has no terminal function.
- SUB 0x1A Substitute

  Has no terminal function.
- ESC 0x1B Escape

  Defines the beginning of an escape or control sequence. Subsequent codes are saved until a final character (0x30 through 0x7F) is found. If the ESC was followed immediately by a '[' code, subsequent codes are saved until a final character (0x40 through 0x7F) is found. In either case the sequence is then interpreted as defined in "Escape and Control Code Sequences" on page 3-83. If undefined, the entire sequence including the ESC and final character is ignored on receipt from the virtual machine. Keyed sequences are returned to the virtual machine, but ignored when echoed. Also, if the sequence is invalid, an error Device Status Report is sent to the virtual machine.

  If ESC is keyed and not to be echoed, the ESC is returned to the virtual machine along with any other data. The VTMP does not interpret the ESC as the first character of a function string in this case. Subsequent received characters will be processed without any effect from the ESC.

  If ESC is to be echoed, the VTMP interprets the ESC as the beginning of an ANSI function string. Subsequent received characters are then included as part of the function until the end of the sequence is reached or the sequence is determined to be invalid. Invalid sequences are

returned to the virtual machine as a regular, unechoed character sequence. The character that makes the sequence invalid is treated normally with respect to echo and break mapping.

- SS4 0x1C Single Shift 4

  Indicates that the following 8-bit code is interpreted as belonging to the upper half of the P2 code page.

- SS3 0x1D Single Shift 3

  Indicates that the following 8-bit code is interpreted as belonging to the lower half of the P2 code page.

- SS2 0x1E Single Shift 2

  Indicates that the following 8-bit code is interpreted as belonging to the upper half of the P1 code page.

- SS1 0x1F Single Shift 1

  Indicates that the following 8-bit code is interpreted as belonging to the lower half of the P1 code page.

Escape and Control Code Sequences

This section defines the code points and effects on the virtual terminal for escape sequences and control sequences which are recognized in KSR data. All escape sequences must begin with the ESC code, 0x1B. The end of an escape sequence is recognized by a code in the range 0x30 through 0x7F. All control sequences must begin with the ESC code, 0x1B, followed by a '[', 0x5B. The end of a control sequence is recognized by a code in the range 0x40 through 0x7F. All escape or control sequences not defined below are ignored on receipt from the virtual machine. Invalid sequences return an error Device Status Report to the virtual machine. Escape or control sequences of more than 34 codes are considered invalid on receipt of the 35th code. The next code is not considered a part of that sequence. Also, numeric parameters in control sequences contain no more than three digits. The numeric value of the parameter may be incorrect if more than three digits are used, and the numeric value never exceeds 255. Controls affect a virtual terminal's presentation space (PS) and its related cursor (pointer into the PS). The presentation space is a logical array of display symbols, N columns by M lines.

The ASCII escape and control code sequences in ANSI mode are described below. A line introducing each control gives its mnemonic, its code sequence, and its function.

Please note that all control sequences flow from the operating system to the VTMP, with the following exceptions:

- KSI (keyboard status information) and PFK (PF key report) flow from the VTMP to the operating system.
- VTA (virtual terminal addressability), VTL (virtual terminal locator), and VTR (virtual terminal raw keyboard input) flow from the VTMP to the application by way of the MOM ring buffer only.

- DSR (device status report) and CPR (cursor position report) can flow in either direction. The meaning of these controls depends on the direction.

The escape and control sequences are defined as follows:

- CBT ESC [PN Z Cursor Back Tab

  Moves the cursor back the number of horizontal tab stops specified by 'PN'. Tab stops are always set at the first and last columns of each line. If the cursor is already in the first column of a line and WRAP mode is set, the cursor moves to the last column. If AUTONL is also set, the cursor moves to the last column of the previous line. In this case, if the cursor is already on the first row of the PS, it moves to the last row.

- CHA ESC [PN G Cursor Horizontal Absolute

  Moves the cursor to the column specified by 'PN', unless the column exceeds the PS width. If the column exceeds the PS width, the cursor moves to the PS column farthest to the right.

- CHT ESC [PN I Cursor Horizontal Tab

  Moves the cursor position to the next tab specified by 'PN'. If the cursor is already on the last tab of a line and WRAP mode is set, the cursor is moved to the first tab. If AUTONL is also set, the cursor moves to the next line.

- CTC ESC [PS W Cursor Tab Stop Control

  0 = Set a horizontal tab at cursor
  1 = Set a vertical tab at cursor
  2 = Clear a horizontal tab at cursor
  3 = Clear a vertical tab at cursor
  4 = Clear all horizontal tabs on line
  5 = Clear all horizontal tabs
  6 = Clear all vertical tabs.

  Sets or clears one or more tabulation stops according to the parameter specified. Tab stops on the first or last column cannot be cleared. When horizontal tab stops are set or cleared, the number of lines affected is all (if Tabulation Stop Mode is reset) or one (if Tabulation Stop Mode is set). This control does not change the position of characters already in the presentation space.

- CNL ESC [PN E Cursor Next Line

  Moves the cursor down the number of lines specified by 'PN', and over to the first position of that line. If the cursor was already on the bottom PS line and WRAP mode is not set, it is positioned at the beginning of that line. If WRAP mode is set, the cursor wraps from the bottom line to the top PS line.

- CPL ESC [PN F Cursor Preceding Line

  Moves the cursor back the number of lines specified by 'PN', and over to the first position of that line. If the cursor was already on the top PS line and WRAP mode is not set, the cursor is positioned at the beginning of that line. If WRAP mode is set, the cursor wraps from the top line to the bottom line of the PS.

- CPR ESC [PN ; PN R Cursor Position Report

  Reports the current cursor position. The first numeric parameter is the line number, and the second is the column. Line and column values are sent to the virtual machine as information. However, if the information is received by the virtual terminal, it is treated as a CUP control.

- CUB ESC [PN D Cursor Backward

  Moves the cursor backward on the line the specified number of columns. If this cursor movement exceeds the left PS boundary and WRAP mode is not set, the cursor stops at the leftmost PS position. If WRAP mode is set, the cursor wraps from the leftmost column to the rightmost column of the preceding PS line. In WRAP mode the cursor also wraps from the home to the rightmost bottom position of the PS.

- CUD ESC [PN B Cursor Down

  Moves the cursor down the number of lines specified by 'PN'. If this cursor movement exceeds the bottom PS boundary and WRAP mode is not set, the cursor stops on the last PS line. If WRAP mode is set, the cursor wraps from the bottom line to the top line of the PS.

- CUF ESC [PN C Cursor Forward

  Moves the cursor forward on the line the specified number of columns. If this cursor movement exceeds the right PS boundary and WRAP mode is not set, the cursor stops at the rightmost PS position. If WRAP mode is set, the cursor wraps from the rightmost column to the leftmost column of the following line in the PS. In WRAP mode, the cursor also wraps from rightmost bottom position to the home position of the PS.

- CUP ESC [PN ; PN H Cursor Position

  Moves the cursor to the line specified by the first parameter, and to the column specified by the second parameter. If this movement crosses a PS boundary, the cursor stops at the PS boundary.

- CUU ESC [PN A Cursor up

  Moves the cursor up the specified number of lines. If this cursor movement exceeds the top PS boundary and WRAP mode is not set, the cursor stops on the first PS line. If WRAP mode is set, the cursor wraps from the top line to the bottom line in the PS.

- CVT ESC [PN Y Cursor Vertical Tab

  Moves the cursor down the number of vertical tab stops specified. Tab stops are assumed at the top and bottom PS lines. If there are not enough vertical tab stops in the PS and WRAP mode is not set, the cursor stops on the last line in the PS. If WRAP mode is set, the cursor wraps from the bottom line to the top line of the PS.

- DCH ESC [PN P Delete Character

  Deletes the cursor character and the following 'PN-1' characters on the cursored line. The characters following the deleted characters on the line are shifted backward to overlay the deleted character positions and the line is cleared from the end of the line to the edge of the presentation space. If the number of characters to be deleted exceeds the number of columns

from the cursor to the PS right boundary, then all the characters from the cursor to the PS boundary are cleared.

- DL ESC [PN M Delete Line

  Deletes the line and the 'PN-1' following lines in the PS. The lines following the deleted lines are scrolled up 'PN' lines and 'PN' blanks lines are placed at the bottom of the PS. If there are less than 'PN' lines from the cursored line to the bottom of the PS, the cursored line and all the following PS lines are replaced with empty lines.

- DSR ESC [PN N Device Status Report/Request

  6 Request Cursor Position Report

  13 Error Report.

  A request cursor position report sends a cursor position report from the virtual terminal to the virtual machine.

  An error report is sent from the virtual terminal to the virtual machine when the virtual terminal receives an invalid control sequence.

  Error reports are private reports which conform to the ANSI standard for private parameters.

- DMI ESC ' Disable Manual Input

  This control, when received in an output data stream, causes keyboard input to this terminal to be ignored. DMI has no terminal function when received from the keyboard.

- EMI ESC b Enable Manual Input

  This control, when received in an output data stream or from the keyboard, restarts keyboard input recognition if it was previously disabled with a DMI escape sequence.

- EA ESC [0 0 Erase to end of area

  ESC [1 0 Erase from start of area

  ESC [2 0 Erase all of area.

  This control is treated like an EL control sequence. The final character for the EA control is a capital O, not a zero.

- ED ESC [0 J Erase to end of display

  ESC [1 J Erase from start of display

  ESC [2 J Erase all of display.

  Erases certain characters within the PS. Erased characters are replaced with empty spaces. Erase to end of display erases the cursored character and all following characters in the PS. Erase from start of display erases the first character of first line and the following characters up to and including the cursored character. Erase all of display erases all the characters on the PS.

- EF ESC [0 N Erase to end of Field
  ESC [1 N Erase from start of Field
  ESC [2 N Erase all of Field.

  Erases certain characters between horizontal tab stops. Erased characters are replaced with empty spaces. Erase to end of field erases the cursored character and all following characters prior to the next tab stop. Erase from start of field erases the character at the tab stop preceding the cursor an the following characters up to and including the cursored character. Erase all of field erases the character at the tab stop preceding the cursor, and the following characters up to and including the character at the tab stop following the cursor. Tab stops are assumed at the first and last columns of the PS when executing this control.

- EL ESC [0 K Erase to end of Line
  ESC [1 K Erase from start of Line
  ESC [2 K Erase all of Line.

  Erases certain characters within a line. Erased characters are replaced with empty spaces. Erase to end of Line erases the cursored character and all following characters on the line. Erase from start of Line erases the first character of first line and the following characters up to and including the cursored character. Erase all of Line erases all the characters on the line.

- ECH ESC [PN X Erase Character

  Erases the cursored character and the following 'PN-1' characters on that line. Erased characters are replaced with empty spaces. If there are less than 'PN' characters from the cursor to the PS right boundary, then the cursored character and all the following characters on the line are replaced empty spaces.

- HTS ESC H Horizontal Tab Stop

  Sets a horizontal tab stop at the current horizontal position. If TSM is set, then the tab stop will apply only to this line. If TSM is reset, then the tab stop applies to all PS lines. This control does not change the positioning of characters already in the presentation space.

- HVP ESC [PN ; PN f Horizontal and Vertical Position

  Moves the cursor to the line specified by the first parameter, and to the column specified by the second parameter. If this movement would cross a PS boundary, the cursor stops at the current PS boundary.

- ICH ESC [PN @ Insert Character

  Inserts 'PN' empty spaces in front of the cursored character. The string of characters starting with the cursored character and ending with last character of the line are shifted 'PN' columns to the right. Characters shifted past the PS right boundary are lost. The cursor does not move.

- IL ESC [PN L Insert Line

  The contents of the cursored line and all lines below it are scrolled down 'PN' lines and 'PN' blank lines are inserted. The absolute screen position of the cursor is not affected.

- IND ESC D Index

  Moves cursor down one line. If the cursor was already on the bottom line of the PS, then the top line is lost, the other lines move up one line, and a blank line becomes the new bottom line.

- NEL ESC E Next Line

  Moves the cursor to the first position of the following line. If the cursor was already on the bottom line of the PS, then the top line is lost, the other lines move up one, and a blank line becomes the new bottom line.

- KSI ESC [PS p Keyboard Status Information

  The virtual terminal generates this control whenever HOSTS and XLATKBD are set and the status of the keyboard changes. Each selective parameter is the ASCII-encoded decimal value of a keyboard status byte. For example, if the keyboard has two status bytes, the control sequence will be **ESC [xxx;yyy p**, where **xxx** is the value of the high-order byte and **yyy** is the value of the low-order byte. This is a private control that conforms to the ANSI standards for private control sequences. The virtual terminal display handler ignores this sequence whether it is received from the virtual machine or echoed.

- PFK ESC [PN q PF Key Report

  The control sequence is sent by the virtual terminal to the virtual machine when a program function key (PFK) code is received from the keyboard. The parameter 'PN' is a PF key number from 1 to 255. This is a private control which conforms to the ANSI standards for private control sequences. This sequence is ignored by the virtual terminal display handler whether received from the virtual machine or echoed.

- RCP ESC [u Restore Cursor Position

  Moves the cursor to the position saved by the last SCP control. If no SCP has been received, then the cursor position is set to the first character of the first line. This is a private control that conforms to the ANSI standards for private controls. This control has no terminal function when received from the keyboard.

- RI ESC L Reverse Index

  Moves the cursor up one line, unless the cursor is already on the PS top line. In that case, if WRAP mode is not set, then the cursor does not move. If WRAP mode is set, the cursor moves to the bottom line of the PS. The column position does not change.

- RIS ESC c Reset to Initial State

  Resets the virtual terminal to the state of a newly-opened virtual terminal: erases all PS data, places the cursor at the home position, resets graphic rendition to normal attributes, default font and color, and sets tab stops, modes, keyboard map, character maps and echo maps to their default values.

- RM ESC [PS l Reset Mode

  2 0 LNM - Line Feed - New Line Mode (default = 1)
  4 IRM - Insert Replace Mode (default = 0)

1 2 SRM - Send Receive Mode (set echo on) (default = 0)
1 8 TSM - Tabulation Stop Mode (default = 0)
? 2 1 CNM - Carriage Return - New Line Mode (default = 0)
? 7 AUTONL - Wrap character to following line when end of current line reached (default = 1)

Resets the modes specified in the parameter string. Multiple parameters must be separated by semi-colons. The modes that can be reset are listed above with the appropriate parameter code. All other mode parameters are ignored.

TSM mode determines whether horizontal tabs apply identically to all line (TSM reset) or uniquely to each line on which they are set (TSM set).

- SCP ESC [s Save Cursor Position

Saves the current cursor position. Any previously saved cursor position is lost. The cursor can be restored to this position with an RCP control. This is a private control that conforms to the ANSI standards for private controls. This control has no terminal function when received from the keyboard.

- SD ESC [PN T Scroll Down

Moves all the PS lines down 'PN' lines. The bottom 'PN' lines are lost, and 'PN' empty lines are put at the top of the PS. Physical cursor position does not change as a result of the scroll.

- SL ESC [PN SP @ Scroll Left

Moves all the PS characters 'PN' column positions to the left. The characters in the 'PN' leftmost PS columns are lost, and empty spaces are put in the rightmost 'PN' columns of all lines. Physical cursor position does not change as a result of the scroll.

- SR ESC [PN SP A Scroll Right

Moves all the PS characters 'PN' column positions to the right. The characters in the 'PN' rightmost PS columns are lost, and empty spaces are put in the leftmost 'PN' columns of all lines. Physical cursor position does not change as a result of the scroll.

- SU ESC [PN S Scroll Up

Moves all the PS lines up 'PN' lines. The top 'PN' lines are lost, and 'PN' empty lines are put at the bottom of the PS. The physical cursor position does not change as a result of the scroll.

- SGR ESC [PS m Set Graphic Rendition

0 Normal (none of attributes 1-9)
1 Bold or Bright
4 Underscore
5 Slow Blink
7 Negative (reverse image)
8 Cancelled On (invisible: set to background color)
10 Primary Font
11 First Alternate Font

12 Second Alternate Font
13 Third Alternate Font
14 Fourth Alternate Font
15 Fifth Alternate Font
16 Sixth Alternate Font
17 Seventh Alternate Font
30 Color palette entry 0 foreground (default to black)
31 Color palette entry 1 foreground (default to red)
32 Color palette entry 2 foreground (default to green)
33 Color palette entry 3 foreground (default to yellow)
34 Color palette entry 4 foreground (default to blue)
35 Color palette entry 5 foreground (default to magenta)
36 Color palette entry 6 foreground (default to cyan)
37 Color palette entry 7 foreground (default to white)
40 Color palette entry 0 background (default to black)
41 Color palette entry 1 background (default to red)
42 Color palette entry 2 background (default to green)
43 Color palette entry 3 background (default to yellow)
44 Color palette entry 4 background (default to blue)
45 Color palette entry 5 background (default to magenta)
46 Color palette entry 6 background (default to cyan)
47 Color palette entry 7 background (default to white)
90 Color palette entry 8 foreground (default to gray)
91 Color palette entry 9 foreground (default to light red)
92 Color palette entry 10 foreground (default to light green)
93 Color palette entry 11 foreground (default to brown)
94 Color palette entry 12 foreground (default to light blue)
95 Color palette entry 13 foreground (default to light magenta)
96 Color palette entry 14 foreground (default to light cyan)
97 Color palette entry 15 foreground (default to high-intensity white)
100 Color palette entry 8 background (default to gray)
101 Color palette entry 9 background (default to light red)
102 Color palette entry 10 background (default to light green)
103 Color palette entry 11 background (default to brown)
104 Color palette entry 12 background (default to light blue)
105 Color palette entry 13 background (default to light magenta)
106 Color palette entry 14 background (default to light cyan)
107 Color palette entry 15 background (default to high intensity white).

Causes the subsequent characters received in the data stream or from the keyboard to have the display attributes specified by the parameter string. Any parameter not listed above is ignored.

The attributes corresponding to parameters 1 through 9 are cumulative. For example, if you specify 'underscore' and then issue another SGR to specify 'blink', subsequent characters will be underscored and blink. To reset one of these attributes, you have to specify 'normal' and then reinstate the parameters you want. Multiple parameters are processed in the order listed.

Whether the characters actually have the requested attributes on the display depends on the capabilities of the physical display device used by the virtual terminal.

Note that, unlike loading new fonts with the Change Fonts command ("Change Fonts" on page 3-95), switching between loaded fonts with the SGR sequence causes no data loss.

Nondisplayable characters do not exist in this system.

- SGOA ESC (f Set G0 Character Set

  SGOB ESC , f Set G0 Character Set (Alternate form)

  : Unique One (User-defined)
  ; Unique Two (User-defined)
  < P0 (Display Symbols 32-255)
  = P1 (Display Symbols 256-479)
  > P2 (Display Symbols 480-703)
  ? User1 (Display Symbols 704-927)
  @ User2 (Display Symbols 928-1023)

  Designates the set of characters to use as the G0 set, when the G0 set is invoked by SI. The default G0 set is the 224-character PCASCII set ('<') in 8-bit PCASCII mode. Unique One and Unique Two may have unique definitions for each virtual terminal. When a virtual terminal is opened, these two sets are equivalent to '<'. The 224-character sets ('<', '@', '=', '>', and '?') are available for use.

- SG1A ESC) f Set G1 Character Set

  SG1B ESC - f Set G1 Character Set (Alternate)

  : Unique One (User-defined)
  ; Unique Two (User-defined)
  < P0 (Display Symbols 32-255)
  = P1 (Display Symbols 256-479)
  > P2 (Display Symbols 480-703)
  ? User1 (Display Symbols 704-927)
  @ User2 (Display Symbols 928-1023)

  Designates the set of characters to use as the G1 set, when the G1 set is invoked by SO. The default G1 set is the 224-character PCASCII set ('<') for 8-bit PCASCII mode. Unique One and Unique Two may have unique definitions for each virtual terminal. When a virtual terminal is opened, these two sets are equivalent to '<'. The 224-character sets ('<', '@', '=', '>', and '?') are available for use.

- SM ESC [PS h Set Mode

  2 0 LNM - Line Feed - New Line Mode (default = 1)
  4 IRM - Insert Replace Mode (default = 0)
  1 2 SRM - Send Receive Mode (set echo off) (default = 0)
  1 8 TSM - Tabulation Stop Mode (default = 0)
  ? 2 1 CNM - Carriage Return - New Line Mode (default = 0)

? 7 AUTONL - Wrap character to following line when end of current line reached (default = 1)

Sets the modes specified in the parameter string. Multiple parameters must be separated by semi-colons. The modes that can be set are listed above with the appropriate parameter code. All other mode parameters are ignored.

Note: Some display devices do not have unique display font symbols for the single-byte control codes.

SRM mode affects translated keyboard input handling. If SRM mode is set, translated keyboard input is never echoed by the virtual terminal, but is immediately returned to the virtual machine.

TSM mode determines whether horizontal tabs apply to all lines identically (TSM reset) or if horizontal tabs apply uniquely to each line on which they are set (TSM set).

- TBC ESC [PS g Tabulation Clear

  0 Horizontal tab at cursor column
  1 Vertical tab at cursored line
  2 Horizontal tabs on line
  3 Horizontal tabs in PS
  4 Vertical tabs in PS

  Clears tabulation stops specified by the parameters. Horizontal tab changes affect only the cursored line if TSM is set, and horizontal tab changes affect all lines if TSM is reset. Any parameters not listed above are ignored. This control does not change the positioning of characters already in the presentation space.

- VTA ESC [r Virtual Terminal Addressability

  This private control sequence precedes a binary header and associated data that provide status information on the 5081 adapter.

- VTD ESC [x Virtual Terminal Data

  This private control sequence precedes a binary header and associated data. The block of data may be in formats other than ASCII character coded data.

  Note: Some of the data following the VTD sequence is binary, not coded character data.

- VTL ESC [y Virtual Terminal Locator Information

  This private control sequence precedes binary locator input data when a buffer is used to send input from the VTMP to a virtual machine.

  Note: Some of the data following the VTD sequence is binary, not coded character data.

- VTR ESC [w Virtual Terminal Raw Keyboard Input

  This private control sequence precedes binary raw keyboard input data when a buffer is used to send input from the VTMP to a virtual machine and XLATKBD is reset.

  Note: Some of the data following the VTD sequence is binary, not coded character data.

- VTS ESC I Vertical Tab Stop

  Sets a vertical tab stop at the cursored line. This control does not change the positioning of characters already in the presentation space.

KSR Protocol Mode Definition

Protocol modes determine how the VTMP interprets coded data, and how the VTMP translates and returns input data within a virtual terminal. A single word of protocol mode data follows the parameter data.

Note that two bits control each mode; one bit indicates whether to use the current mode setting. If the current setting is not used, the second bit indicates which specific mode value should be used. The VTMP sets mode bits to the default value when a virtual terminal is first opened. You can change these defaults during configuration.

If WRAP mode is set, the cursor is allowed to wrap around the presentation space.

If HOSTPC mode is set, the virtual machine may receive locator position reports.

If XLATKBD mode is set, the VTMP converts data received from the keyboard to ASCII codes. If XLATKBD is reset, the VTMP returns key position, scan code and status.

KSR protocol modes are defined by bytes 0-3 of the minor data area.

The following section tells you how to identify KSR protocol modes and change default values.

Protocol Mode

HOSTS Specifies whether to return keyboard shift status changes

Bit 3 = 0: use current value
Bit 3 = 1: specify return in bit 19
  Bit 19 = 0: do not return keyboard status change (default)
  Bit 19 = 1: return private ANSI sequence with change information

XLATKBD Specifies whether to translate keyboard input to scan codes

Bit 7 = 0: use current value
Bit 7 = 1: specify translation in bit 23
  Bit 23 = 0: return key position, status, scan code
  Bit 23 = 1: XLATKBD (default)

HOSTPC Specifies whether to send locator input to host

Bit 8 = 0: use current value
Bit 8 = 1: specify whether to send input in bit 24
  Bit 24 = 0: do not return locator input to host (default)
  Bit 24 = 1: return locator input to host

WRAP Specifies whether text cursor wraps in presentation space

Bit 9 = 0: use current value
Bit 9 = 1: specify wrap in bit 25
Bit 25 = 0: do not wrap cursor
Bit 25 = 1: wrap cursor (default)

HOSTLPFK Specifies whether to send input from the lighted programmable function keys to the host

Bit 10 = 0: use current value
Bit 10 = 1: specify whether to send input in bit 26
Bit 26 = 0: do not return LPF key input (default)
Bit 26 = 1: return LPF key input

HOSTDIALS Specifies whether to send input from the dials to the host

Bit 11 = 0: use current value
Bit 11 = 1: specify whether to send input in bit 27
Bit 27 = 0: do not return dials input (default)
Bit 27 = 1: return dials input

HOSTS mode specifies whether to report keyboard status changes to the virtual machine. If HOSTS is set, the keyboard status information is returned to the virtual machine in a private ANSI control, KSI.

Note: If the virtual machine wants to ensure against keyboard interaction between SVCs in multiple-SVC output operations, the virtual machine should disable the keyboard (DMI). Also, the virtual machine can request a cursor position report before sending display data. The reported cursor position can then be restored after the data is sent so operator cursor positioning is not lost.

Character Set Definition

You can alter the ASCII character set-to-display code (font) mapping of a virtual terminal. For each virtual terminal, the VTMP maintains character set mapping tables for 2 unique user-definable character sets. These sets contain 256 10-bit display symbol codes, and are identified by the final characters ';' or ':' of the SG0 or SG1 controls.

To modify the user-definable character sets, take any of the sets above as a source, modify all or part of it and place the newly-defined set into either Unique One or Unique Two. Until they are first modified, both user-definable sets are identical to the PCASCII standard set. Either set may be modified by designating its ANSI final character (';' or ':'). Any or all of the positions in a set may be replaced with any of the 10-bit display symbols. The remap is defined to begin at position 00. Therefore, to remap a single position, you may either send the entire character map with only one position deviating from the current map, or you can send a map up to and including the one remapped character. All subsequent remappings to the same unique set will be applied over the existing map.The actual graphical representation of a display symbol is dependent on the hardware in use.

Note: Data is kept in display symbol form in the virtual terminal. A datastream returned in the query ASCII codes acknowledgement uses the standard character set definitions, not Unique One or Two.

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0	User-definable character set : = Unique one ; = Unique two
1	Reserved
2-3	10-bit display symbol code This field may be repeated up to 256 times.

Set KSR Color Palette

This command specifies the color you want to associate with certain display adapters. The default color palette attempts to match the ANSI 3.64 palette. This may not be possible for all devices.

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0-1	Number of entries in the palette
2-5	Adapter-specific settings of the first entry in the color palette. These settings must be repeated for each entry of the color palette corresponding to the display adapter.

See "Acknowledge Query Physical Device" on page 3-65 for information on the maximum number of entries for the active display.

Change Fonts

This data changes the assignment of physical display fonts to the SGR font attributes. When a virtual terminal is first opened, and whenever it is changed to a different physical display, this assignment is made to the first font in the list of configured fonts. The VTMP initially tries to select a font that results in a presentation space of 80 columns by 25 rows. Of the fonts that meet this criteria, the first font with a normal appearance (not italic) is chosen. If no fonts meet this criteria, the first font that can be displayed on the particular device is selected. All alternate fonts will be initialized to this chosen ID.

Note that if you change fonts, you will lose the data currently in the presentation space, and the cursor reverts to the double underscore and is placed at the home position (first column, first row). Therefore, IBM recommends that, if you want control of the fonts, you should explicitly set the fonts you want when you open a terminal or change a display.

All eight font attributes must be assigned to fonts of the same size. If any of the requested IDs do not exist, no changes will be made. The Change Font request will be denied and a return code will be

sent in the OPSB indicating that one or more non-existent IDs was requested. Otherwise, if all eight requested fonts are not the same size, no changes will be made and a return code in the OPSB indicates that one or more different sized IDs was requested.

If the Change Fonts request is accepted and the installed fonts are a different size than the previous fonts, the presentation space size is adjusted to the number of rows and columns that fit on the physical display screen for the new font size.

Note: The **Query Physical Display** command is used to determine the font IDs.

The following field appears at the start of minor type data and is defined as follows:

Bytes Value

0-1 Physical font ID of primary font attribute

2-3 Physical font ID of first alternate font attribute

4-5 Physical font ID of second alternate font attribute

6-7 Physical font ID of third alternate font attribute

8-9 Physical font ID of fourth alternate font attribute

10-11 Physical font ID of fifth alternate font attribute

12-13 Physical font ID of sixth alternate font attribute

14-15 Physical font ID of seventh alternate font attribute.

Text cursor representation

The text cursor representation field determines how the cursor is manifested in the display screen.

This field appears at the start of minor type data and is defined as follows:

Bytes Value

0-1 Text cursor shape

0 = No cursor
1 = Single underscore
2 = Double underscore
3 = Illuminated character cell (lower half)
4 = Double mid-character line
5 = Illuminated character cell.

Query ASCII Codes and Attributes

This data determines how you define a block of characters in the presentation space to query. You will receive attribute and character set information on the characters within the queried block in a query ASCII codes and attributes acknowledgement.

Note: The **VT Query SVC** path must be used for this command.

This field appears at the start of minor type data and is defined as follows:

Bytes	Value
0	X upper left coordinate (first column of the block)
1	Y upper left coordinate (first row in the block)
2	X lower right coordinate (last column in the block)
3	Y lower right coordinate (last row in the block)

Acknowledge Query ASCII Codes and Attributes

The data for this minor type is an ASCII data stream that contains character codes from the queried block. Character set changes and changes to character attributes are indicated with SGR and SG0 control sequences. A line feed control is returned after the last character code in each line of the queried block.

Note: The returned attributes may be only a subset of the original attributes specified for query. The subset in this case is those attributes actually supported by the physical device and associated with a character.

The following field appears at the start of minor type data and is defined as follows:

Bytes	Value
0-n	ASCII data stream

This field includes all ASCII data currently associated with the input buffer.

MOM Major Data Type

A major type field of 0x05 identifies MOM data. Each MOM minor data type has a different structure and interpretation.

Monitored mode provides applications with a direct output path to display hardware and a shortened input path for the keyboard and locator. The minor types associated with MOM data are summarized in the following table and described in subsequent sections.

Value	Minor Type
0x00	ASCII Codes (assumed in absence of VTD and header)
0x01	MOM protocol mode definition
0x02	Screen Request and Input Ring Addressability
0x03	Screen Release Acknowledgement
0x04	DMA Move
0xA0	Screen Grant Interrupt
0xA1	Screen Release Interrupt
0xA2	Application Terminated Interrupt

ASCII Codes

Monitored mode allows two ASCII controls to control the keyboard input device. These controls, enable manual input (EMI) and disable manual input (DMI), are the only ASCII controls monitored mode recognizes. All other codes are ignored if received.

MOM Protocol Mode Definition

Protocol modes determine how the VTMP returns input data from the input devices. The protocol command is the first command used in monitored mode; it informs the VTMP that the application has selected monitored, as opposed to KSR, mode.

When you run applications in monitored mode you have several options as far as protocols are concerned. The protocol decisions you have to make in MOM mode include:

- Whether to enable or disable locator input
- Whether to place the keyboard in native or translated mode
- Whether to enable or disable dials input to the host

- Whether to enable or disable LPF key input to the host.
- Whether to return status information from display adapters.
- Whether to limit the use of the MOM input ring buffer only for display adapter-generated status information.

Aside from the bits described above, all other bits in the 32-bit MOM protocol definition word must be set equal to zero. If you try to set any other bits in the protocol definition word, the VTMP rejects the command and does not allow the application to enter monitored mode.

This field appears at the start of minor type data and is defined as follows:

Protocol Mode

XLATKBD Specifies whether to translate keyboard input to scan codes

Bit 7 = 0: use current value
Bit 7 = 1: specify translation in bit 23
Bit 23 = 0: return key position, status, scan code
Bit 23 = 1: XLATKBD (default)

HOSTPC Specifies whether to send locator input to host

Bit 8 = 0: use current value
Bit 8 = 1: specify whether to send input in bit 24
Bit 24 = 0: do not return locator input to host (default)
Bit 24 = 1: return locator input to host

HOSTLPFK Specifies whether to send input from the lighted programmable function keys to the host

Bit 10 = 0: use current value
Bit 10 = 1: specify whether to send input in bit 26
Bit 26 = 0: do not return LPF key input (default)
Bit 26 = 1: return LPF key input

HOSTDIALS Specifies whether to send input from the dials to the host

Bit 11 = 0: use current value
Bit 11 = 1: specify whether to send input in bit 27
Bit 27 = 0: do not return dials input (default)
Bit 27 = 1: return dials input

HOSTDINTR Specifies whether to send display adapter status information to the host

Bit 12 = 0: use current value
Bit 12 = 1: specify whether to send status information in bit 28
Bit 28 = 0: do not return status information to the host (default)
Bit 28 = 1: return status information to the host

HOSTDINTRONLY Specifies whether to use a MOM input ring buffer only for returning display adapter status information to the host

Bit 13 = 0: use current value
Bit 13 = 1: specify whether to limit input ring buffer in bit 29
Bit 29 = 0: do not limit use of input ring buffer (default)
Bit 27 = 1: limit use of input ring buffer to display adapter status information.

Screen Request and Input Ring Addressability

When a MOM application requests ownership of the display hardware, the VTMP receives the address and size of the associated input buffer. The input buffer ring contains status and control information, as well as input device events. The ring, which has a width of four bytes, must be aligned on a word boundary within the application virtual address.

If the minor type data is not included with the screen request VTD, all keyboard and locator data are returned through the normal unsolicited interrupt flow. The length of this VTD includes the entire length of the ring buffer, as well.

The input buffer ring, regardless of size, consists of two partitions. The first partition is a 32-byte field reserved for status and control information. The remaining bytes comprise the second partition, which is the actual ring buffer for keyboard and locator input entries. The second partition serves as an event queue which receives event reports from the input devices. Input from either a keyboard or locator enters the queue in the order they are received. The single event queue guarantees first-in first-out queueing.

Figure 3-27 illustrates the input ring.

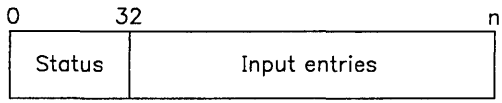


Figure 3-27. Input Ring Format

The status partition (bytes 0 through 31) is defined in Figure 3-28 on page 3-101.

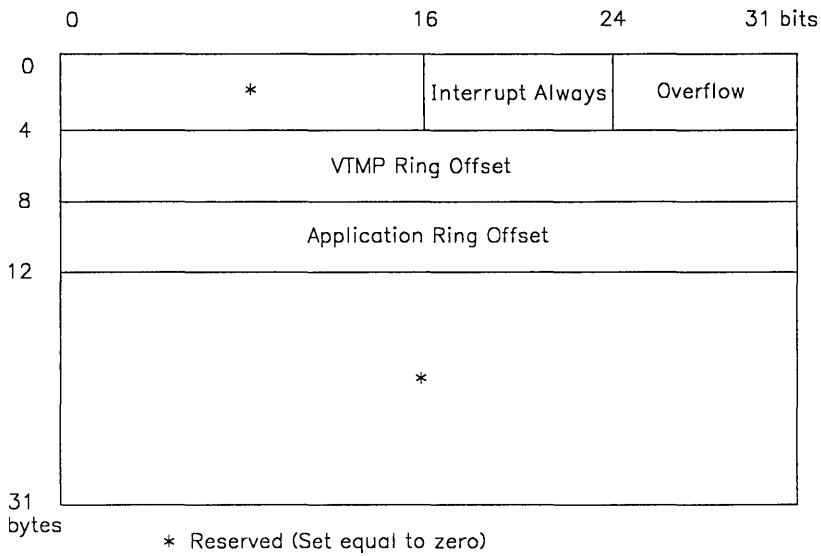


Figure 3-28. Status Partition of Input Ring

The following fields appear at the start of the minor type data request VTD and ring buffer and are defined as follows:

Bytes Value

0-1 Length of input ring in bytes

2-5 Offset to the input buffer ring (offset from start of minor data)

Start of buffer ring (must start on a word boundary)

0-1 Reserved

2 Interrupt always

3 Overflow

4-7 Ring offset for the VTMP

8-11 Ring offset for the application

12-31 Reserved

32 First data byte in the ring

N Last data byte in the ring.

After you set up a monitored mode virtual terminal and the desired protocol, you issue this command to request control of the display hardware. The VTMP confirms that your application has control of the display with a screen grant short unsolicited interrupt (ID = 20).

Warning: The Screen Request and Input Ring Addressability command must be issued by way of the VT Output SVC if the application is defining the monitored mode input ring.

The status partition contains four major fields. Bytes 0 and 1 and 12 through 31 are reserved (set equal to zero). The remaining fields within the first partition are defined as follows:

- Interrupt always

  Byte 2 of the status partition determines whether the VTMP should cause a signal to be sent to the application each time data is placed into the ring. Note that the VTMP resets this byte after each input.
- Overflow

  Byte 3 of the status partition determines whether the input buffer ring can accommodate more input information. A value of 0xFF indicates an overflow; 0x00 indicates normal operation.
- VTMP Ring Offset

  The next four bytes of this partition represent the offset into the input ring to which the VTMP enqueues keyboard and locator input. This offset starts from the beginning of the ring, so the absolute minimum value for the VTMP offset is 32 bytes.
- Application Ring Offset

  The next four bytes of the status partition indicate the offset into the input ring from which the application reads keyboard and locator information from the event queue. This offset also starts from the beginning of the input ring, so the minimum value for this offset is 32 bytes.

The VTMP sets the overflow flag when the VTMP offset into the input ring plus the length of the keyboard or locator information the VTMP wants to write exceed the offset from which the application reads. For example, suppose the VTMP offset is set at 38 bytes and the application offset is set at 45 bytes. The VTMP receives 12 bytes of input data from the keyboard to place in the input ring. Because the sum of these two fields exceeds the application offset (38 + 12 > 45), the buffer is 'overflowed' and the VTMP sets the overflow byte flag.

When an overflow condition exists for a buffer, the input data is not stored in the input ring, but is lost. The VTMP will store no more data in the input ring until the application clears the overflow flag. Therefore, the VTMP must detect a potential overflow **before** placing data in the input ring. Both the application and the VTMP can read its own or the other's offset value. The VTMP or application can prevent a potential overflow condition by modifying its offset, but should never modify the offset of its counterpart.

Full and Empty Queues

Two other conditions describe the state of the input queue. When the VTMP offset is one byte less than the application offset, the queue is said to be 'full'. When the VTMP offset is equal to the application offset, the queue is said to be 'empty'.

When a virtual terminal is initialized to run in the monitored mode, the application sets the overflow byte equal to 0x00 and both offset values equal to 32 (0x20). As normal operation progresses, the offsets contain different values. The application offset typically falls behind the VTMP offset as the application tries to keep up with input received by the VTMP.

An application can clear the input queue, therefore, by setting its offset equal to the VTMP offset. Both offsets do not have to start from 32, and an 'empty' queue may contain data unrelated to the present application.

If the queue is empty when the VTMP places input into the ring, the VTMP generates a short unsolicited interrupt (ID = 24) to the operating system terminal device driver. The interrupt does not contain the actual input data, because the VTMP places the input data into the input ring as usual after sending the interrupt.

Keyboard Input

When the keyboard is in translate mode, the VTMP checks all keystrokes against the break map before placing the character or function into the input ring. If the corresponding break map bit is off, the VTMP stores the keystroke in the ring and generates no unsolicited interrupt unless the queue was empty. If the corresponding break map bit is on or the input ring has not been defined, the VTMP sends the input content (character or function) in an unsolicited interrupt to the virtual terminal driver in the operating system.

If a keystroke is translated into a string and none of the corresponding break map bits are on, the VTMP places the entire string in the input ring. If a keystroke is translated into a string with one or more of the corresponding break map bits on, the VTMP does not place the entire string into the input ring. Instead, the VTMP sends the string to the MOM application by way of the operating system's terminal device driver. You should be careful, therefore, when you set the break map for MOM applications. Because excessive use of the break map in this case degrades performance, you should use it sparingly with MOM applications.

Regardless of the input format (character, string, or function), the input buffer (if defined) consists of ANSI characters, controls, control sequences, and private control sequences. If the keyboard is set in the native mode, the VTMP stores all keystrokes in the input ring as fixed-length entries consisting of scan code, position code and shift status preceded by the private control 0x1B5B77 which denotes virtual terminal raw (VTR) keyboard input. All break, echo and translate maps are inoperative. Format of the scan codes, position codes and shift status are provided in a short unsolicited interrupt.

An application that changes the keyboard mode in midstream is entirely responsible for keeping track of the potential mixture of ASCII characters and controls with raw scan codes in the input ring.

Locator Input

Relative locator input consists of relative horizontal and vertical distances in addition to the status of the mouse buttons and the time stamp information. Distances are reported only when either the horizontal or vertical threshold is exceeded or when the status of a button changes. When the status of a button changes, the VTMP reports the change, as well as the relative distance change (if any).

Absolute locator input consists of absolute horizontal and vertical positions in addition to the status of the four buttons on the tablet. Positions are reported only when movement occurs inside the valid reporting area of the tablet (that area bounded by the no-input zone, if any). Positions are also reported when the status of a tablet button changes. In the latter case, absolute positions are reported, regardless of whether the sensor device is within the valid reporting area.

All relative locator input is stored in the buffer ring as fixed-length entries consisting of horizontal and vertical deltas, time stamp data and status information. This information is preceded by the private control sequence virtual terminal locator (VTL - 0x1B5B79). The last byte of relative locator data is set to zero.

All absolute locator input is handled in like manner, except that the horizontal and vertical values are absolute positions rather than deltas from a previous value. The last byte of absolute locator data is set to one.

The format of locator input is described in "VT Unsolicited Interrupt" on page 3-45.

If the MOM input buffer is not defined, locator input is returned by way of the normal unsolicited interrupt flow.

You can shut off locator event reports by manipulating the HOSTPC bit in the protocol mode definition. Under normal circumstances, the VTMP stops accepting locator input only if the buffer overflows.

The MOM input buffer cannot be used if the minor type data is omitted.

Screen Release Acknowledgement

When the system has to suspend ownership of the display hardware by an application, the VTMP issues a screen release unsolicited interrupt (ID =21). The application must send this acknowledgement message to the VTMP before the time out interval expires. If the application does not respond with the screen release acknowledgement, the VTMP issues the Application Terminate Interrupt (minor type 0xA2) to remove the uncooperative application. For more information on the details of these unsolicited interrupt types, see "VT Unsolicited Interrupt" on page 3-45.

DMA Move

When an application is using DMA to move data between its space and system DMA space, this minor type is used to change the pointer to the data. The following fields come immediately after the major type (0x05) and the DMA move minor type:

- 2 bytes for the segment ID of the DMA data
- 4 bytes indicating the source address
- 4 bytes indicating the length of the DMA data
- 4 bytes indicating the target address.

If the source address falls in the range 0xE0800000 through 0xE08FFFFF, the source of the data is system DMA space. If the source address is outside this range, the source of the data is application DMA space.

Note that the DMA move minor type causes the pointer to DMA data to be changed. The DMA data range is defined as the nearest 2K boundary before and after the DMA area. Therefore, aligning and moving DMA data on 2K boundaries eliminates the overhead of copying partial pages that are not included in the DMA data range.

Keyboard Translate Table

The following table lists each key position and provides the scan code, assignment, and returned string for each of the four valid key states (base, shift, control, alternate) for the 101-key Enhanced Personal Computer keyboard (in U.S. English). Note that the alternate graphics state is not included in this table. Figure 3-29 shows all the key positions on the standard 101-key RT PC keyboard. For information on the 102-key and 106-key keyboards and the 15 available keyboard layouts, see *IBM RT PC Keyboard Description and Character Reference.*

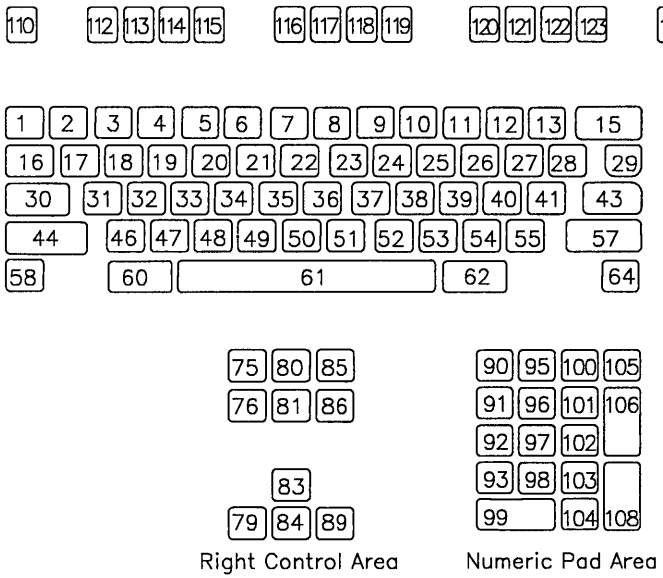


Figure 3-29. Position Codes for Remapping a 101-Key Keyboard

Note that the scan code for a particular key does not change when the key is remapped. The scan code is returned to the virtual machine when running in untranslated mode.

Figure 3-33 on page 3-137 shows the characters available from the P0 code page.

Figure 3-34 on page 3-138 shows the characters available from the P1 code page.

Figure 3-35 on page 3-139 shows the characters available from the P2 code page.

The following key positions in the specified states are not redefinable on the 101-key, 102-key, or 106-key keyboards. Their functions are predefined by the VTRM or VTMP.

Key Position and State	Function
	Caps lock
44 (all)	Left shift
57 (all)	Right shift
58 (all)	Control shift
64 alternate	Next virtual terminal
64 shift	Last virtual terminal
64 control	Command virtual terminal
90 base	Num lock
90 shift	Num lock.

The following key positions cannot be remapped on the 101-key or 102-key keyboard:

Key Position and State	Function
30 (all)	Caps lock
60 (all)	Alternate
62 (all)	Alternate graphics

The following key positions cannot be remapped on the 106-key keyboard only:

Key Position and State	Function
60 (all)	Caps lock
62 (all)	Right alternate

Also note that the following positions do not appear on the 101-key RT PC keyboard and cannot be remapped: 14, 42, 45, 56, 59, 63, 65-74, 77-78, 82, 84, 87-88, 94, 107, 109 and 111.

The keyboard translate table begins on the following page.

Key Posn.	Key State	Scan Code	Assignment	Returned String
1	base	0x0E	grave accent	0x60
	shift		tilde	0x7e
	ctrl		PFK 57	ESC [0 5 7 q
	alt		PFK 115	ESC [1 1 5 q
2	base	0x16	1 one	0x31
	shift		! exclamation pt.	0x21
	crtl		PFK 49	ESC [0 4 9 q
	alt		PFK 58	ESC [0 5 8 q
3	base	0x1E	2 two	0x32
	shift		@ at sign	0x40
	crtl		NUL null	0x00
	alt		PFK 59	ESC [0 5 9 q
4	base	0x26	3 three	0x33
	shift		# number sign	0x23
	ctrl		PFK 50	ESC [0 5 0 q
	alt		PFK 60	ESC [0 6 0 q
5	base	0x25	4 four	0x34
	shift		$ dollar sign	0x24
	ctrl		PFK 51	ESC [0 5 1 q
	alt		PFK 61	ESC [0 6 1 q
6	base	0x2E	5 five	0x35
	shift		% percent sign	0x25
	ctrl		PFK 52	ESC [0 5 2 q
	alt		PFK 62	ESC [0 6 2 q
7	base	0x36	6 six	0x36
	shift		¬ circumflex	0x5e
	ctrl		RS	0x1e
	alt		PFK 63	ESC [0 6 3 q
8	base	0x3D	7 seven	0x37
	shift		& ampersand	0x26
	ctrl		PFK 53	ESC [0 5 3 q
	alt		PFK 64	ESC [0 6 4 q

9	base	0x3E	8 eight	0x38
	shift		* asterisk	0x2a
	ctrl		PFK 54	ESC [0 5 4 q
	alt		PFK 65	ESC [0 6 5 q
10	base	0x46	9 nine	0x39
	shift		(left paren.	0x28
	ctrl		PFK 55	ESC [0 5 5 q
	alt		PFK 66	ESC [0 6 6 q
11	base	0x45	0 zero	0x30
	shift		) right paren.	0x29
	ctrl		PFK 56	ESC [0 5 6 q
	alt		PFK 67	ESC [0 6 7 q
12	base	0x4E	- hyphen	0x2d
	shift		_ underscore	0x5f
	ctrl		US unit separator	0x1f
	alt		PFK 68	ESC [0 6 8 q
13	base	0x55	= equal sign	0x3d
	shift		+ plus sign	0x2b
	ctrl		PFK 69	ESC [0 6 9 q
	alt		PFK 70	ESC [0 7 0 q
15	base	0x66	BS back space	0x08
	shift		BS back space	0x08
	ctrl		DEL delete	0x7f
	alt		PFK 71	ESC [0 7 1 q
16	base	0x0D	Horiz. tab	0x09
	shift		Cursor back tab	ESC [Z
	ctrl		PFK 72	ESC [0 7 2 q
	alt		PFK 73	ESC [0 7 3 q
17	base	0x15	q lowercase q	0x71
	shift		Q capital Q	0x51
	ctrl		DC1 device ctrl	0x11
	alt		PFK 74	ESC [0 7 4 q
18	base	0x1D	w lowercase w	0x77
	shift		W capital W	0x57
	ctrl		ETB end trans blk	0x17
	alt		PFK 75	ESC [0 7 5 q
19	base	0x24	e lowercase e	0x65
	shift		E capital E	0x45
	ctrl		ENQ enquiry	0x05
	alt		PFK 76	ESC [0 7 6 q

20	base	0x2D	r lowercase r	0x72
	shift		R capital R	0x52
	ctrl		DC2 Device ctrl 2	0x12
	alt		PFK 77	ESC [0 7 7 q
21	base	0x2C	t lowercase t	0x74
	shift		T capital T	0x54
	ctrl		DC4 Device ctrl 4	0x14
	alt		PFK 78	ESC [0 7 8 q
22	base	0x35	y lowercase y	0x79
	shift		Y capital Y	0x59
	ctrl		EM End of media	0x19
	alt		PFK 79	ESC [0 7 9 q
23	base	0x3C	u lowercase u	0x75
	shift		U capital U	0x55
	ctrl		NAK not ack.	0x15
	alt		PFK 80	ESC [0 8 0 q
24	base	0x43	i lowercase i	0x69
	shift		I capital I	0x49
	ctrl		HT horizontal tab	0x09
	alt		PFK 81	ESC [0 8 1 q
25	base	0x44	o lowercase o	0x6f
	shift		O capital O	0x4f
	ctrl		SI shift in	0x0f
	alt		PFK 82	ESC [0 8 2 q
26	base	0x4D	p lowercase p	0x70
	shift		P capital P	0x50
	ctrl		DLE data link enable	0x10
	alt		PFK 83	ESC [0 8 3 q
27	base	0x54	[left bracket	0x5b
	shift		{ left brace	0x7b
	ctrl		ESC escape	0x1b
	alt		PFK 84	ESC [0 8 4 q
28	base	0x5B	] right bracket	0x5d
	shift		} right brace	0x7d
	ctrl		GS group separator	0x1d
	alt		PFK 85	ESC [0 8 5 q
29	base	0x5C	\ reverse slash	0x5c
	shift		| pipe symbol	0x7c
	ctrl		FS file separator	0x1c
	alt		PFK 86	ESC [0 8 6 q

30	base	0x14	caps lock	not returned
	shift		caps lock	not returned
	ctrl		caps lock	not returned
	alt		caps lock	not returned
31	base	0x1C	a lowercase a	0x61
	shift		A capital A	0x41
	ctrl		SOH start hdr.	0x01
	alt		PFK 87	ESC [0 8 7 q
32	base	0x1B	s lowercase s	0x73
	shift		S capital S	0x53
	ctrl		DC3 device ctrl 3	0x13
	alt		PFK 88	ESC [0 8 8 q
33	base	0x23	d lowercase d	0x64
	shift		D capital D	0x44
	ctrl		EOT End of trans.	0x04
	alt		PFK 89	ESQ [0 8 9 q
34	base	0x2B	f lowercase f	0x66
	shift		F capital F	0x46
	ctrl		ACK Acknowledge	0x06
	alt		PFK 90	ESQ [0 9 0 q
35	base	0x34	g lowercase g	0x67
	shift		G capital G	0x47
	ctrl		BEL bell	0x07
	alt		PFK 91	ESQ [0 9 1 q
36	base	0x33	h lowercase h	0x68
	shift		H capital H	0x48
	ctrl		BS backspace	0x08
	alt		PFK 92	ESC [0 9 2 q
37	base	0x3B	j lowercase j	0x6a
	shift		J capital J	0x4a
	ctrl		LF line feed	0x0a
	alt		PFK 93	ESC [0 9 3 q
38	base	0x42	k lowercase k	0x6b
	shift		K capital K	0x4b
	ctrl		VT vertical tab	0x0b
	alt		PFK 94	ESC [0 9 4 q

39	base	0x4B	l lowercase l	0x6c
	shift		L capital L	0x4c
	ctrl		FF form feed	0x0c
	alt		PFK 95	ESC [0 9 5 q
40	base	0x4C	; semicolon	0x3b
	shift		: colon	0x3a
	ctrl		PFK 96	ESC [0 9 6 q
	alt		PFK 97	ESC [0 9 7 q
41	base	0x52	' quote/apostro.	0x27
	shift		" double quote	0x22
	ctrl		PFK 98	ESC [0 9 8 q
	alt		PFK 99	ESC [0 9 9 q
43	base	0x5A	CR carriage return	0x0d
	shift		CR carriage return	0x0d
	ctrl		CR carriage return	0x0d
	alt		PFK 100	ESC [1 0 0 q
44	base	0x12	shift (left)	not returned
	shift		shift (left)	not returned
	ctrl		shift (left)	not returned
	alt		shift (left)	not returned
46	base	0x1A	z lowercase z	0x7a
	shift		Z capital Z	0x5a
	ctrl		SUB substitute char	0x1a
	alt		PFK 101	ESC [1 0 1 q
47	base	0x22	x lowercase x	0x78
	shift		X capital X	0x58
	ctrl		CAN cancel	0x18
	alt		PFK 102	ESC [1 0 2 q
48	base	0x21	c lowercase c	0x63
	shift		C capital C	0x43
	ctrl		ETX End of text	0x03
	alt		PFK 103	ESC [1 0 3 q
49	base	0x2A	v lowercase v	0x76
	shift		V capital V	0x56
	ctrl		SYN synch idle	0x16
	alt		PFK 104	ESC [1 0 4 q
50	base	0x32	b lowercase b	0x62
	shift		B capital B	0x42
	ctrl		STX start of text	0x02
	alt		PFK 105	ESC [1 0 5 q

51	base	0x31	n lowercase n	0x6e
	shift		N capital N	0x4e
	ctrl		SO shift out	0x0e
	alt		PFK 106	ESC [1 0 6 q
52	base	0x3A	m lowercase m	0x6d
	shift		M capital M	0x4d
	ctrl		CR carriage return	0x0d
	alt		PFK 107	ESC [1 0 7 q
53	base	0x41	, comma	0x2c
	shift		< less than sign	0x3c
	ctrl		PFK 108	ESC [1 0 8 q
	alt		PFK 109	ESC [1 0 9 q
54	base	0x49	. period	0x2e
	shift		> greater than	0x3e
	ctrl		PFK 110	ESC [1 1 0 q
	alt		PFK 111	ESC [1 1 1 q
55	base	0x4A	/ slash	0x2f
	shift		? question mark	0x3f
	ctrl		PFK 112	ESC [1 1 2 q
	alt		PFK 113	ESC [1 1 3 q
57	base	0x59	shift (right)	not returned
	shift		shift (right)	not returned
	ctrl		shift (right)	not returned
	alt		shift (right)	not returned
58	base	0x11	control	not returned
	shift		control	not returned
	ctrl		control	not returned
	alt		control	not returned
60	base	0x19	alternate shift	not returned
	shift		alternate shift	not returned
	ctrl		alterbate shift	not returned
	alt		alternate shift	not returned
61	base	0x29	SP space	0x20
	shift		SP space	0x20
	ctrl		SP space	0x20
	alt		SP space	0x20
62	base	0x39	alternate shift	not returned
	shift		none	not returned
	ctrl		none	not returned
	alt		alternate shift	not returned

64	base	0x58	PFK 114	ESC [1 1 4 q
	shift		Last virtual terminal	Last virtual terminal
	ctrl		Command virtual terminal	Command virtual terminal
	alt		Next virtual terminal	Next virtual terminal
75	base	0x67	PFK 139 INS toggle	ESC [1 3 9 q
	shift		PFK 139 INS toggle	ESC [1 3 9 q
	ctrl		PFK 140	ESC [1 4 0 q
	alt		PFK 141	ESC [1 4 1 q
76	base	0x64	DCH delete char	ESC [P
	shift		DCH delete char	ESC [P
	ctrl		PFK 142	ESC [1 4 2 q
	alt		DL delete line	ESC [M
79	base	0x61	CUB cursor back	ESC [D
	shift		PFK 158	ESC [1 5 8 q
	ctrl		PFK 159	ESC [1 5 9 q
	alt		PFK 160	ESC [1 6 0 q
80	base	0x6E	HOME	ESC [H
	shift		PFK 143	ESC [1 4 3 q
	ctrl		PFK 144	ESC [1 4 4 q
	alt		PFK 145	ESC [1 4 5 q
81	base	0x65	PFK 146	ESC [1 4 6 q
	shift		PFK 147	ESC [1 4 7 q
	ctrl		PFK 148	ESC [1 4 8 q
	alt		PFK 149	ESC [1 4 9 q
83	base	0x63	CUU cursor up	ESC [A
	shift		PFK 161	ESC [1 6 1 q
	ctrl		PFK 162	ESC [1 6 2 q
	alt		PFK 163	ESC [1 6 3 q
84	base	0x60	CUD cursor down	ESC [B
	shift		PFK 164	ESC [1 6 4 q
	ctrl		PFK 165	ESC [1 6 5 q
	alt		PFK 166	ESC [1 6 6 q
85	base	0x6F	PFK 150	ESC [1 5 0 q
	shift		PFK 151	ESC [1 5 1 q
	ctrl		PFK 152	ESC [1 5 2 q
	alt		PFK 153	ESC [1 5 3 q
86	base	0x6D	PFK 154	ESC [1 5 4 q
	shift		PFK 155	ESC [1 5 5 q
	ctrl		PFK 156	ESC [1 5 6 q
	alt		PFK 157	ESC [1 5 7 q

89	base	0x6A	CUF cursor forward	ESC [C
	shift		PFK 167	ESC [1 6 7 q
	ctrl		PFK 168	ESC [1 6 8 q
	alt		PFK 169	ESC [1 6 9 q
90	base	0x76	NUM LOCK	not returned
	shift		NUM LOCK	not returned
	ctrl		DC3 device ctrl 3	0x13
	alt		PFK 170	ESC [1 7 0 q
91	base	0x6C	┌	0xda
	shift		7 seven	0x37
	ctrl		PFK 172	ESC [1 7 2 q
	alt*		alt+num entry	return at alt break
92	base	0x6B	├	0xc3
	shift		4 four	0x34
	ctrl		PFK 174	ESC [1 7 4 q
	alt*		alt+num entry	return at alt break
93	base	0x69	└	0xc0
	shift		1 one	0x31
	ctrl		PFK 176	ESC [1 7 6 q
	alt*		alt+num entry	return at alt break
95	base	0x77	/ slash	0x2f
	shift		/ slash	0x2f
	ctrl		PFK 179	ESC [1 7 9 q
	alt		PFK 180	ESC [1 8 0 q
96	base	0x75	┬	0xc2
	shift		8 eight	0x38
	ctrl		PFK 182	ESC [1 8 2 q
	alt*		alt+num entry	return at alt break
97	base	0x73	┼	0xc5
	shift		5 five	0x35
	ctrl		PFK 184	ESC [1 8 4 q
	alt*		alt+num entry	return at alt break
98	base	0x72	┴	0xc1
	shift		2 two	0x32
	ctrl		PFK 186	ESC [1 8 6 q
	alt*		alt+num entry	return at alt break
99	base	0x70	|	0xb3
	shift		0 zero	0x30
	ctrl		PFK 178	ESC [1 7 8 q
	alt*		alt+num entry	return at alt break

100	base	0x7E	* asterisk	0x2a
	shift		* asterisk	0x2a
	ctrl		PFK 187	ESC [1 8 7 q
	alt		PFK 188	ESC [1 8 8 q
101	base	0x7D	┐	0xbf
	shift		9 nine	0x39
	ctrl		PFK 190	ESC [1 9 0 q
	alt*		alt + num entry	return at alt break
102	base	0x74	┤	0xb4
	shift		6 six	0x36
	ctrl		PFK 192	ESC [1 9 2 q
	alt*		alt + num entry	return at alt break
103	base	0x7A	┘	0xd9
	shift		3 three	0x33
	ctrl		PFK 194	ESC [1 9 4 q
	alt*		alt + num entry	return at alt break
104	base	0x71	− horizontal line	0xc4
	shift		. period	0x2e
	ctrl		PFK 196	ESC [1 9 6 q
	alt		PFK 197	ESC [1 9 7 q
105	base	0x84	- Hyphen (minus)	0x2d
	shift		- hyphen (minus)	0x2d
	ctrl		PFK 198	ESC [1 9 8 q
	alt		PFK 199	ESC [1 9 9 q
106	base	0x7C	+ plus	0x2b
	shift		+ plus	0x2b
	ctrl		PFK 200	ESC [2 0 0 q
	alt		PFK 201	ESC [2 0 1 q
108	base	0x79	CR carriage return	0x0d
	shift		CR carriage return	0x0d
	ctrl		CR carriage return	0x0d
	alt		PFK 100	ESC [1 0 0 q
110	base	0x08	ESC	0x1b
	shift		PFK 120	ESC [1 2 0 q
	ctrl		PFK 121	ESC [1 2 1 q
	alt		PFK 122	ESC [1 2 2 q
112	base	0x07	PFK 1	ESC [0 0 1 q
	shift		PFK 13	ESC [0 1 3 q
	ctrl		PFK 25	ESC [0 2 5 q
	alt		PFK 37	ESC [0 3 7 q

113	base	0x0F	PFK 2	ESC [0 0 2 q
	shift		PFK 14	ESC [0 1 4 q
	ctrl		PFK 26	ESC [0 2 6 q
	alt		PFK 38	ESC [0 3 8 q
114	base	0x17	PFK 3	ESC [0 0 3 q
	shift		PFK 15	ESC [0 1 5 q
	ctrl		PFK 27	ESC [0 2 7 q
	alt		PFK 39	ESC [0 3 9 q
115	base	0x1F	PFK 4	ESC [0 0 4 q
	shift		PFK 16	ESC [0 1 6 q
	ctrl		PFK 28	ESC [0 2 8 q
	alt		PFK 40	ESC [0 4 0 q
116	base	0x27	PFK 5	ESC [0 0 5 q
	shift		PFK 17	ESC [0 1 7 q
	ctrl		PFK 29	ESC [0 2 9 q
	alt		PFK 41	ESC [0 4 1 q
117	base	0x2F	PFK 6	ESC [0 0 6 q
	shift		PFK 18	ESC [0 1 8 q
	ctrl		PFK 30	ESC [0 3 0 q
	alt		PFK 42	ESC [0 4 2 q
118	base	0x37	PFK 7	ESC [0 0 7 q
	shift		PFK 19	ESC [0 1 9 q
	ctrl		PFK 31	ESC [0 3 1 q
	alt		PFK 43	ESC [0 4 3 q
119	base	0x3F	PFK 8	ESC [0 0 8 q
	shift		PFK 20	ESC [0 2 0 q
	ctrl		PFK 32	ESC [0 3 2 q
	alt		PFK 44	ESC [0 4 4 q
120	base	0x47	PFK 9	ESC [0 0 9 q
	shift		PFK 21	ESC [0 2 1 q
	ctrl		PFK 33	ESC [0 3 3 q
	alt		PFK 45	ESC [0 4 5 q
121	base	0x4F	PFK 10	ESC [0 1 0 q
	shift		PFK 22	ESC [0 2 2 q
	ctrl		PFK 34	ESC [0 3 4 q
	alt		PFK 46	ESC [0 4 6 q
122	base	0x56	PFK 11	ESC [0 1 1 q
	shift		PFK 23	ESC [0 2 3 q
	ctrl		PFK 35	ESC [0 3 5 q
	alt		PFK 47	ESC [0 4 7 q

123	base	0x5E	PFK 12	ESC [0 1 2 q
	shift		PFK 24	ESC [0 2 4 q
	ctrl		PFK 36	ESC [0 3 6 q
	alt		PFK 48	ESC [0 4 8 q
124	base	0x57	PFK 209	ESC [2 0 9 q
	shift		PFK 210	ESC [2 1 0 q
	ctrl		PFK 211	ESC [2 1 1 q
	alt		PFK 212	ESC [2 1 2 q
125	base	0x5F	PFK 213	ESC [2 1 3 q
	shift		PFK 214	ESC [2 1 4 q
	ctrl		PFK 215	ESC [2 1 5 q
	alt		PFK 216	ESC [2 1 6 q
126	base	0x62	PFK 217	ESC [2 1 7 q
	shift		PFK 218	ESC [2 1 8 q
	ctrl		DEL	0x7F
	alt		DEL	0x7F

Display Symbols by Code Page

The graphic characters available from code pages 0, 1, and 2 are described in the following tables. The organization of the code pages is illustrated in Figure 3-33 on page 3-137, Figure 3-34 on page 3-138, and Figure 3-35 on page 3-139. Positions 0 through 31 (0x00 through 0x1F) contain single-byte controls and are common to all three code pages.

Position Number	Character	Code Page/ Code Point	Internal Code
32	Space	P0-32 (0x20)	0x20
33	! Exclamation Point	P0-33 (0x21)	0x21
34	" Double Quote	P0-34 (0x22)	0x22
35	# Number Sign	P0-35 (0x23)	0x23
36	$ Dollar Sign	P0-36 (0x24)	0x24
37	% Percent Sign	P0-37 (0x25)	0x25
38	& Ampersand	P0-38 (0x26)	0x26
39	' Apostrophe, Acute Accent	P0-39 (0x27)	0x27
40	(Left Parenthesis	P0-40 (0x28)	0x28
41	) Right Parenthesis	P0-41 (0x29)	0x29
42	* Asterisk	P0-42 (0x2A)	0x2A
43	+ Plus Sign	P0-43 (0x2B)	0x2B
44	, Comma	P0-44 (0x2C)	0x2C
45	- Hyphen, Minus Sign	P0-45 (0x2D)	0x2D
46	. Period	P0-46 (0x2E)	0x2E
47	/ Slash	P0-47 (0x2F)	0x2F
48	0 Zero	P0-48 (0x30)	0x30
49	1 One	P0-49 (0x31)	0x31
50	2 Two	P0-50 (0x32)	0x32
51	3 Three	P0-51 (0x33)	0x33
52	4 Four	P0-52 (0x34)	0x34
53	5 Five	P0-53 (0x35)	0x35
54	6 Six	P0-54 (0x36)	0x36
55	7 Seven	P0-55 (0x37)	0x37
56	8 Eight	P0-56 (0x38)	0x38
57	9 Nine	P0-57 (0x39)	0x39
58	: Colon	P0-58 (0x3A)	0x3A
59	; Semicolon	P0-59 (0x3B)	0x3B
60	< Less Than Sign	P0-60 (0x3C)	0x3C
61	= Equal Sign	P0-61 (0x3D)	0x3D
62	> Greater Than Sign	P0-62 (0x3E)	0x3E
63	? Question Mark	P0-63 (0x3F)	0x3F
64	@ At Sign	P0-64 (0x40)	0x40
65	A a Uppercase	P0-65 (0x41)	0x41
66	B b Uppercase	P0-66 (0x42)	0x42
67	C c Uppercase	P0-67 (0x43)	0x43
68	D d Uppercase	P0-68 (0x44)	0x44

Figure 3-30 (Part 1 of 7). Code Page P0

Position Number	Character		Code Page/ Code Point	Internal Code
69	E	e Uppercase	P0-69 (0x45)	0x45
70	F	f Uppercase	P0-70 (0x46)	0x46
71	G	g Uppercase	P0-71 (0x47)	0x47
72	H	h Uppercase	P0-72 (0x48)	0x48
73	I	i Uppercase	P0-73 (0x49)	0x49
74	J	j Uppercase	P0-74 (0x4A)	0x4A
75	K	k Uppercase	P0-75 (0x4B)	0x4B
76	L	l Uppercase	P0-76 (0x4C)	0x4C
77	M	m Uppercase	P0-77 (0x4D)	0x4D
78	N	n Uppercase	P0-78 (0x4E)	0x4E
79	O	o Uppercase	P0-79 (0x4F)	0x4F
80	P	p Uppercase	P0-80 (0x50)	0x50
81	Q	q Uppercase	P0-81 (0x51)	0x51
82	R	r Uppercase	P0-82 (0x52)	0x52
83	S	s Uppercase	P0-83 (0x53)	0x53
84	T	t Uppercase	P0-84 (0x54)	0x54
85	U	u Uppercase	P0-85 (0x55)	0x55
86	V	v Uppercase	P0-86 (0x56)	0x56
87	W	w Uppercase	P0-87 (0x57)	0x57
88	X	x Uppercase	P0-88 (0x58)	0x58
89	Y	y Uppercase	P0-89 (0x59)	0x59
90	Z	z Uppercase	P0-90 (0x5A)	0x5A
91	[	Left Bracket	P0-91 (0x5B)	0x5B
92	\	Reverse Slash	P0-92 (0x5C)	0x5C
93	]	Right Bracket	P0-93 (0x5D)	0x5D
94	^	Circumflex Accent, Up Arrow	P0-94 (0x5E)	0x5E
95	_	Underline, Low Line	P0-95 (0x5F)	0x5F
96	`	Grave Accent, Left Single Quote	P0-96 (0x60)	0x60
97	a	a Lowercase	P0-97 (0x61)	0x61
98	b	b Lowercase	P0-98 (0x62)	0x62
99	c	c Lowercase	P0-99 (0x63)	0x63
100	d	d Lowercase	P0-100 (0x64)	0x64
101	e	e Lowercase	P0-101 (0x65)	0x65
102	f	f Lowercase	P0-102 (0x66)	0x66
103	g	g Lowercase	P0-103 (0x67)	0x67
104	h	h Lowercase	P0-104 (0x68)	0x68
105	i	i Lowercase	P0-105 (0x69)	0x69

Figure 3-30 (Part 2 of 7). Code Page P0

Position Number	Character		Code Page/ Code Point	Internal Code
106	j	j Lowercase	P0-106 (0x6A)	0x6A
107	k	k Lowercase	P0-107 (0x6B)	0x6B
108	l	l Lowercase	P0-108 (0x6C)	0x6C
109	m	m Lowercase	P0-109 (0x6D)	0x6D
110	n	n Lowercase	P0-110 (0x6E)	0x6E
111	o	o Lowercase	P0-111 (0x6F)	0x6F
112	p	p Lowercase	P0-112 (0x70)	0x70
113	q	q Lowercase	P0-113 (0x71)	0x71
114	r	r Lowercase	P0-114 (0x72)	0x72
115	s	s Lowercase	P0-115 (0x73)	0x73
116	t	t Lowercase	P0-116 (0x74)	0x74
117	u	u Lowercase	P0-117 (0x75)	0x75
118	v	v Lowercase	P0-118 (0x76)	0x76
119	w	w Lowercase	P0-119 (0x77)	0x77
120	x	x Lowercase	P0-120 (0x78)	0x78
121	y	y Lowercase	P0-121 (0x79)	0x79
122	z	z Lowercase	P0-122 (0x7A)	0x7A
123	{	Left Brace	P0-123 (0x7B)	0x7B
124	|	Logical OR	P0-124 (0x7C)	0x7C
125	}	Right Brace	P0-125 (0x7D)	0x7D
126	~	Tilde Accent	P0-126 (0x7E)	0x7E
127	Δ	Del	P0-127 (0x7F)	0x7F
128	Ç	c Cedilla Capital	P0-128 (0x80)	0x80
129	ü	u Umlaut Small	P0-129 (0x81)	0x81
130	é	e Acute Small	P0-130 (0x82)	0x82
131	â	a Circumflex Small	P0-131 (0x83)	0x83
132	ä	a Umlaut Small	P0-132 (0x84)	0x84
133	à	a Grave Small	P0-133 (0x85)	0x85
134	å	a Overcircle Small	P0-134 (0x86)	0x86
135	ç	c Cedilla Small	P0-135 (0x87)	0x87
136	ê	e Circumflex Small	P0-136 (0x88)	0x88
137	ë	e Umlaut Small	P0-137 (0x89)	0x89
138	è	e Grave Small	P0-138 (0x8A)	0x8A
139	ï	i Umlaut Small	P0-139 (0x8B)	0x8B
140	î	i Circumflex Small	P0-140 (0x8C)	0x8C
141	ì	i Grave Small	P0-141 (0x8D)	0x8D
142	Ä	a Umlaut Capital	P0-142 (0x8E)	0x8E

Figure 3-30 (Part 3 of 7). Code Page P0

Position Number	Character		Code Page/ Code Point	Internal Code
143	Å	a Overcircle Capital	P0-143 (0x8F)	0x8F
144	É	e Acute Capital	P0-144 (0x90)	0x90
145	æ	ae Diphthong Small	P0-145 (0x91)	0x91
146	Æ	ae Diphthong Capital	P0-146 (0x92)	0x92
147	ô	o Circumflex Small	P0-147 (0x93)	0x93
148	ö	o Umlaut Small	P0-148 (0x94)	0x94
149	ò	o Grave Small	P0-149 (0x95)	0x95
150	û	u Circumflex Small	P0-150 (0x96)	0x96
151	ù	u Grave Small	P0-151 (0x97)	0x97
152	ÿ	y Umlaut Small	P0-152 (0x98)	0x98
153	Ö	o Umlaut Capital	P0-153 (0x99)	0x99
154	Ü	u Umlaut Capital	P0-154 (0x9A)	0x9A
155	ø	o Slash Small	P0-155 (0x9B)	0x9B
156	£	English Pound Sign	P0-156 (0x9C)	0x9C
157	Ø	o Slash Capital	P0-157 (0x9D)	0x9D
158	×	Multiplication Sign	P0-158 (0x9E)	0x9E
159	*f*	Florin Sign	P0-159 (0x9F)	0x9F
160	á	a Acute Small	P0-160 (0xA0)	0xA0
161	í	i Acute Small	P0-161 (0xA1)	0xA1
162	ó	o Acute Small	P0-162 (0xA2)	0xA2
163	ú	u Acute Small	P0-163 (0xA3)	0xA3
164	ñ	n Tilde Small	P0-164 (0xA4)	0xA4
165	Ñ	n Tilde Capital	P0-165 (0xA5)	0xA5
166	a	Feminine Sign	P0-166 (0xA6)	0xA6
167	o	Masculine Sign	P0-167 (0xA7)	0xA7
168	¿	Inverted Question Mark	P0-168 (0xA8)	0xA8
169	®	Registered Trademark	P0-169 (0xA9)	0xA9
170	¬	Logical Not	P0-170 (0xAA)	0xAA
171	½	One Half	P0-171 (0xAB)	0xAB
172	¼	One Quarter	P0-172 (0xAC)	0xAC
173	¡	Inverted Exclamation Sign	P0-173 (0xAD)	0xAD
174	«	Left Angle Quotes	P0-174 (0xAE)	0xAE
175	»	Right Angle Quotes	P0-175 (0xAF)	0xAF
176	⁞	Quarter Hashed	P0-176 (0xB0)	0xB0
177	▒	Half Hashed	P0-177 (0xB1)	0xB1
178	▓	Full Hashed	P0-178 (0xB2)	0xB2
179	|	Vertical Bar	P0-179 (0xB3)	0xB3

Figure 3-30 (Part 4 of 7). Code Page P0

Position Number	Character		Code Page/ Code Point	Internal Code
180	┤	Right Side Middle	P0-180 (0xB4)	0xB4
181	Á	a Acute Capital	P0-181 (0xB5)	0xB5
182	Â	a Circumflex Capital	P0-182 (0xB6)	0xB6
183	À	a Grave Capital	P0-183 (0xB7)	0xB7
184	©	Copyright Symbol	P0-184 (0xB8)	0xB8
185	╣	Double Right Side Middle	P0-185 (0xB9)	0xB9
186	║	Double Vertical Bar	P0-186 (0xBA)	0xBA
187	╗	Double Upper Right Corner Box	P0-187 (0xBB)	0xBB
188	╝	Double Lower Right Corner Box	P0-188 (0xBC)	0xBC
189	¢	Cent Sign	P0-189 (0xBD)	0xBD
190	¥	Yen Sign	P0-190 (0xBE)	0xBE
191	┐	Upper Right Corner Box	P0-191 (0xBF)	0xBF
192	└	Lower Left Corner Box	P0-192 (0xC0)	0xC0
193	┴	Bottom Side Middle	P0-193 (0xC1)	0xC1
194	┬	Top Side Middle	P0-194 (0xC2)	0xC2
195	├	Left Side Middle	P0-195 (0xC3)	0xC3
196	─	Center Box Bar	P0-196 (0xC4)	0xC4
197	┼	Intersection	P0-197 (0xC5)	0xC5
198	ã	a Tilde Small	P0-198 (0xC6)	0xC6
199	Ã	a Tilde Capital	P0-199 (0xC7)	0xC7
200	╚	Double Lower Left Corner Box	P0-200 (0xC8)	0xC8
201	╔	Double Upper Left Corner Box	P0-201 (0xC9)	0xC9
202	╩	Double Bottom Side Middle	P0-202 (0xCA)	0xCA
203	╦	Double Top Side Middle	P0-203 (0xCB)	0xCB
204	╠	Double Left Side Middle	P0-204 (0xCC)	0xCC
205	═	Double Center Box Bar	P0-205 (0xCD)	0xCD
206	╬	Double Intersection	P0-206 (0xCE)	0xCE
207	¤	International Currency Symbol	P0-207 (0xCF)	0xCF
208	ð	eth Icelandic Small	P0-208 (0xD0)	0xD0
209	Ð	eth Icelandic Capital	P0-209 (0xD1)	0xD1
210	Ê	e Circumflex Capital	P0-210 (0xD2)	0xD2
211	Ë	e Umlaut Capital	P0-211 (0xD3)	0xD3
212	È	e Grave Capital	P0-212 (0xD4)	0xD4
213	ı	Small i Dotless	P0-213 (0xD5)	0xD5
214	Í	i Acute Capital	P0-214 (0xD6)	0xD6
215	Î	i Circumflex Capital	P0-215 (0xD7)	0xD7
216	Ï	i Umlaut Capital	P0-216 (0xD8)	0xD8

Figure 3-30 (Part 5 of 7). Code Page P0

Position Number	Character		Code Page/ Code Point	Internal Code
217	┘	Lower Right Corner Box	P0-217 (0xD9)	0xD9
218	┌	Upper Left Corner Box	P0-218 (0xDA)	0xDA
219	■	Bright Character Cell	P0-219 (0xDB)	0xDB
220	▄	Bright Character Cell - Lower Half	P0-220 (0xDC)	0xDC
221	¦	Broken Vertical Bar	P0-221 (0xDD)	0xDD
222	Ì	i Grave Capital	P0-222 (0xDE)	0xDE
223	▀	Bright Character Cell - Upper Half	P0-223 (0xDF)	0xDF
224	Ó	o Acute Capital	P0-224 (0xE0)	0xE0
225	β	s Sharp Small	P0-225 (0xE1)	0xE1
226	Ô	o Circumflex Capital	P0-226 (0xE2)	0xE2
227	Ò	o grave capital	P0-227 (0xE3)	0xE3
228	õ	o Tilde Small	P0-228 (0xE4)	0xE4
229	Õ	o Tilde Capital	P0-229 (0xE5)	0xE5
230	µ	Mu Small, Micro Symbol	P0-230 (0xE6)	0xE6
231	þ	Thorn Icelandic Small	P0-231 (0xE7)	0xE7
232	Þ	Thorn Icelandic Capital	P0-232 (0xE8)	0xE8
233	Ú	u Acute Capital	P0-233 (0xE9)	0xE9
234	Û	u Circumflex Capital	P0-234 (0xEA)	0xEA
235	Ù	u Grave Capital	P0-235 (0xEB)	0xEB
236	ý	y Acute Small	P0-236 (0xEC)	0xEC
237	Ý	y Acute Capital	P0-237 (0xED)	0xED
238	¯	Overbar	P0-238 (0xEE)	0xEE
239	´	Acute Accent	P0-239 (0xEF)	0xEF
240	-	Syllable Hyphen	P0-240 (0xF0)	0xF0
241	±	Plus Or Minus Sign	P0-241 (0xF1)	0xF1
242	=	Double Underscore	P0-242 (0xF2)	0xF2
243	¾	Three Fourths	P0-243 (0xF3)	0xF3
244	¶	Paragraph Symbol	P0-244 (0xF4)	0xF4
245	§	Section Symbol	P0-245 (0xF5)	0xF5
246	÷	Division Sign	P0-246 (0xF6)	0xF6
247	¸	Cedilla Accent	P0-247 (0xF7)	0xF7
248	°	Degree Symbol, Overcircle Accent	P0-248 (0xF8)	0xF8
249	¨	Umlaut Accent	P0-249 (0xF9)	0xF9
250	·	Middle Dot, Product Dot	P0-250 (0xFA)	0xFA
251	¹	Superscript 1	P0-251 (0xFB)	0xFB
252	³	Superscript 3	P0-252 (0xFC)	0xFC
253	²	Superscript 2	P0-253 (0xFD)	0xFD

Figure 3-30 (Part 6 of 7). Code Page P0

Position Number	Character		Code Page/ Code Point	Internal Code
254	▌	Vertical Solid Rectangle	P0-254 (0xFE)	0xFE
255		Required Space	P0-255 (0xFF)	0xFF

Figure 3-30 (Part 7 of 7). Code Page P0

Position Number	Character		Code Page/ Code Point	Internal Code
256	•	Spanish Middle Dot	P1-256 (0x20)	0x1FA0
257	☺	Smiling Face	P1-257 (0x21)	0x1FA1
258	☻	Dark Smiling Face	P1-258 (0x22)	0x1FA2
259	♥	Heart	P1-259 (0x23)	0x1FA3
260	♦	Diamond	P1-260 (0x24)	0x1FA4
261	♣	Club	P1-261 (0x25)	0x1FA5
262	♠	Spade	P1-262 (0x26)	0x1FA6
263	•	Bullet	P1-263 (0x27)	0x1FA7
264	◘	Reverse Video Bullet	P1-264 (0x28)	0x1FA8
265	○	Circle	P1-265 (0x29)	0x1FA9
266	◙	Reverse Video Circle	P1-266 (0x2A)	0x1FAA
267	♂	Male Symbol	P1-267 (0x2B)	0x1FAB
268	♀	Female Symbol	P1-268 (0x2C)	0x1FAC
269	♪	Eighth Note	P1-269 (0x2D)	0x1FAD
270	♫	Sixteenth Note	P1-270 (0x2E)	0x1FAE
271	☼	Sun	P1-271 (0x2F)	0x1FAF
272	►	Right Solid Triangle	P1-272 (0x30)	0x1FB0
273	◄	Left Solid Triangle	P1-273 (0x31)	0x1FB1
274	↕	Bidirectional Vertical Arrow	P1-274 (0x32)	0x1FB2
275	‼	Double Exclamation Point	P1-275 (0x33)	0x1FB3
276	¶	Paragraph Symbol	P1-276 (0x34)	0x1FB4
277	§	Section symbol	P1-277 (0x35)	0x1FB5
278	▬	Horizontal Solid Rectangle	P1-278 (0x36)	0x1FB6
279	↨	Underlined Bidirectional Vertical Arrow	P1-279 (0x37)	0x1FB7
280	↑	Up Arrow	P1-280 (0x38)	0x1FB8
281	↓	Down Arrow	P1-281 (0x39)	0x1FB9
282	→	Right Arrow	P1-282 (0x3A)	0x1FBA
283	←	Left Arrow	P1-283 (0x3B)	0x1FBB
284	∟	Diagonally Flipped Logical Not	P1-284 (0x3C)	0x1FBC
285	↔	Bidirectional Horizontal Arrow	P1-285 (0x3D)	0x1FBD
286	▲	Solid Upward Triangle	P1-286 (0x3E)	0x1FBE
287	▼	Solid Downward Triangle	P1-287 (0x3F)	0x1FBF
288	ã	a Tilde Small	P1-288 (0x40)	0x1FC0
289	β	s Sharp Small	P1-289 (0x41)	0x1FC1
290	Â	a Circumflex Capital	P1-290 (0x42)	0x1FC2
291	À	a Grave Capital	P1-291 (0x43)	0x1FC3
292	Á	a Acute Capital	P1-292 (0x44)	0x1FC4

Figure 3-31 (Part 1 of 7). Code Page P1

Position Number	Character		Code Page/ Code Point	Internal Code
293	Ã	a Tilde Capital	P1-293 (0x45)	0x1FC5
294	ø	o Slash Small	P1-294 (0x46)	0x1FC6
295	Ê	e Circumflex Capital	P1-295 (0x47)	0x1FC7
296	Ë	e Umlaut Capital	P1-296 (0x48)	0x1FC8
297	È	e Grave Capital	P1-297 (0x49)	0x1FC9
298	Í	i Acute Capital	P1-298 (0x4A)	0x1FCA
299	Î	i Circumflex Capital	P1-299 (0x4B)	0x1FCB
300	Ï	i Umlaut Capital	P1-300 (0x4C)	0x1FCC
301	Ì	i Grave Capital	P1-301 (0x4D)	0x1FCD
302	Ø	Slashed o Capital	P1-302 (0x4E)	0x1FCE
303	ð	eth Icelandic Small	P1-303 (0x4F)	0x1FCF
304	ý	y Acute Small	P1-304 (0x50)	0x1FD0
305	þ	Thorn Icelandic Small	P1-305 (0x51)	0x1FD1
306	¸	Cedilla Accent	P1-306 (0x52)	0x1FD2
307	¤	International Currency Symbol	P1-307 (0x53)	0x1FD3
308	Ð	eth Icelandic Capital	P1-308 (0x54)	0x1FD4
309	Ý	y Acute Capital	P1-309 (0x55)	0x1FD5
310	Þ	Thorn Icelandic Capital	P1-310 (0x56)	0x1FD6
311	®	Registered Trademark Symbol	P1-311 (0x57)	0x1FD7
312	¾	Three Quarters	P1-312 (0x58)	0x1FD8
313	¯	Overbar Accent, Macron Accent	P1-313 (0x59)	0x1FD9
314	¨	Umlaut Accent	P1-314 (0x5A)	0x1FDA
315	´	Acute Accent	P1-315 (0x5B)	0x1FDB
316	‗	Double Underscore	P1-316 (0x5C)	0x1FDC
317	õ	o Tilde Small	P1-317 (0x5D)	0x1FDD
318	ı	Small i Dotless	P1-318 (0x5E)	0x1FDE
319	Ô	o Circumflex Capital	P1-319 (0x5F)	0x1FDF
320	Ò	o Grave Capital	P1-328 (0x60)	0x1FE0
321	Ó	o Acute Capital	P1-321 (0x61)	0x1FE1
322	Õ	o Tilde Capital	P1-322 (0x62)	0x1FE2
323	³	Superscript 3	P1-323 (0x63)	0x1FE3
324	Û	u Circumflex Capital	P1-324 (0x64)	0x1FE4
325	Ù	u Grave Capital	P1-325 (0x65)	0x1FE5
326	Ú	u Acute Capital	P1-326 (0x66)	0x1FE6
327	ą	a Ogonek Small	P1-327 (0x67)	0x1FE7
328	ě	e Caron Small	P1-328 (0x68)	0x1FE8
329	ň	n Caron Small	P1-329 (0x69)	0x1FE9

Figure 3-31 (Part 2 of 7). Code Page P1

Position Number	Character		Code Page/ Code Point	Internal Code
330	ć	c Acute Small	P1-330 (0x6A)	0x1FEA
331	ę	e Ogonek Small	P1-331 (0x6B)	0x1FEB
332	ů	u Overcircle Small	P1-332 (0x6C)	0x1FEC
333	ď	d Caron Small	P1-333 (0x6D)	0x1FED
334	ĺ	l Acute Small	P1-334 (0x6E)	0x1FEE
335	Ą	a Ogonek Capital	P1-335 (0x6F)	0x1FEF
336	Ě	e Caron Capital	P1-336 (0x70)	0x1FF0
337	Č	c Caron Capital	P1-337 (0x71)	0x1FF1
338	Ć	c Acute Capital	P1-338 (0x72)	0x1FF2
339	ˇ	Caron Accent	P1-339 (0x73)	0x1FF3
340	Ę	e Ogonek Capital	P1-340 (0x74)	0x1FF4
341	Ů	u Overcircle Capital	P1-341 (0x75)	0x1FF5
342	Ď	d Caron Capital	P1-342 (0x76)	0x1FF6
343	Ĺ	l Acute Capital	P1-343 (0x77)	0x1FF7
344	ľ	l Caron Small	P1-344 (0x78)	0x1FF8
345	ň	n Caron Small	P1-345 (0x79)	0x1FF9
346	đ	d Stroke Small	P1-346 (0x7A)	0x1FFA
347	ř	r Caron Small	P1-347 (0x7b)	0x1FFB
348	ś	s Acute Small	P1-348 (0x7C)	0x1FFC
349	°	Overcircle Accent	P1-349 (0x7D)	0x1FFD
350	ł	l Slash Small	P1-350 (0x7E)	0x1FFE
351	ń	n Acute Small	P1-351 (0x7F)	0x1FFF
352	š	s Caron Small	P1-352 (0x80)	0x1E80
353	Ľ	l Caron Capital	P1-353 (0x81)	0x1E81
354	Ň	n Caron Capital	P1-354 (0x82)	0x1E82
355	Ř	r Caron Capital	P1-355 (0x83)	0x1E83
356	Ś	s Acute Capital	P1-356 (0x84)	0x1E84
357	˙	Overdot Accent	P1-357 (0x85)	0x1E85
358	ż	z Overdot Small	P1-358 (0x86)	0x1E86
359	˛	Ogonek Accent	P1-359 (0x87)	0x1E87
360	Ż	z Overdot Capital	P1-360 (0x88)	0x1E88
361	ž	z Caron Small	P1-361 (0x89)	0x1E89
362	ź	z Acute Small	P1-362 (0x8A)	0x1E8A
363	Ž	z Caron Capital	P1-363 (0x8B)	0x1E8B
364	Ź	z Acute Capital	P1-364 (0x8C)	0x1E8C
365	Ł	l Slash Capital	P1-365 (0x8D)	0x1E8D
366	Ń	n Acute Capital	P1-366 (0x8E)	0x1E8E

Figure 3-31 (Part 3 of 7). Code Page P1

Position Number	Character		Code Page/ Code Point	Internal Code
367	Š	s Caron Capital	P1-367 (0x8F)	0x1E8F
368	ť	t Caron Small	P1-368 (0x90)	0x1E90
369	ŕ	r Acute Small	P1-369 (0x91)	0x1E91
370	ő	o Double Acute Small	P1-370 (0x92)	0x1E92
371	ű	u Double Acute Small	P1-371 (0x93)	0x1E93
372	Ţ	t Caron Capital	P1-372 (0x94)	0x1E94
373	Ŕ	r Acute Capital	P1-373 (0x95)	0x1E95
374	Ő	o Double Acute Capital	P1-374 (0x96)	0x1E96
375	Ű	u Double Acute Capital	P1-375 (0x97)	0x1E97
376	ă	a Breve Small	P1-376 (0x98)	0x1E98
377	ğ	g Breve Small	P1-377 (0x99)	0x1E99
378	İ	i Overdot Capital	P1-378 (0x9A)	0x1E9A
379	Ă	a Breve Capital	P1-379 (0x9B)	0x1E9B
380	Ğ	g Breve Capital	P1-380 (0x9C)	0x1E9C
381	˘	Breve Accent	P1-381 (0x9D)	0x1E9D
382	″	Double Acute Accent	P1-382 (0x9E)	0x1E9E
383	ş	s Cedilla Small	P1-383 (0x9F)	0x1E9F
384	ℓ	Liter Symbol	P1-384 (0xA0)	0x1EA0
385	’n	High Comma n Small	P1-385 (0xA1)	0x1EA1
386	Ş	s Cedilla Capital	P1-386 (0xA2)	0x1EA2
387	¯	Macron Accent	P1-387 (0xA3)	0x1EA3
388	ţ	t Cedilla Small	P1-388 (0xA4)	0x1EA4
389	Ţ	t Cedilla Capital	P1-389 (0xA5)	0x1EA5
390	ā	a Macron Small	P1-390 (0xA6)	0x1EA6
391	Ā	a Macron Capital	P1-391 (0xA7)	0x1EA7
392	ĉ	c Circumflex Small	P1-392 (0xA8)	0x1EA8
393	Ĉ	c Circumflex Capital	P1-393 (0xA9)	0x1EA9
394	‵	High Reverse Solidus	P1-394 (0xAA)	0x1EAA
395	ċ	c Overdot Small	P1-396 (0xAB)	0x1EAB
396	Ċ	c Overdot Capital	P1-396 (0xAC)	0x1EAC
397	ė	e Overdot Small	P1-397 (0xAD)	0x1EAD
398	Ė	e Overdot Capital	P1-398 (0xAE)	0x1EAE
399	ē	e Macron Small	P1-399 (0xAF)	0x1EAF
400	Ē	e Macron Capital	P1-400 (0xB0)	0x1EB0
401	ģ	g Acute Small	P1-401 (0xB1)	0x1EB1
402	ĝ	g Circumflex Small	P1-402 (0xB2)	0x1EB2
403	Ĝ	g Circumflex Capital	P1-403 (0xB3)	0x1EB3

Figure 3-31 (Part 4 of 7). Code Page P1

Position Number	Character		Code Page/ Code Point	Internal Code
404	ġ	g Overdot Small	P1-404 (0xB4)	0x1EB4
405	Ġ	g Overdot Capital	P1-405 (0xB5)	0x1EB5
406	Ģ	g Cedilla Capital	P1-406 (0xB6)	0x1EB6
407	ĥ	h Circumflex Small	P1-407 (0xB7)	0x1EB7
408	Ĥ	h Circumflex Capital	P1-408 (0xB8)	0x1EB8
409	ħ	h Stroke Small	P1-409 (0xB9)	0x1EB9
410	Ħ	h Stroke Capital	P1-410 (0xBA)	0x1EBA
411	ĩ	i Tilde Small	P1-411 (0xBB)	0x1EBB
412	Ĩ	i Tilde Capital	P1-412 (0xBC)	0x1EBC
413	ī	i Macron Small	P1-413 (0xBD)	0x1EBD
414	Ī	i Macron Capital	P1-414 (0xBE)	0x1EBE
415	į	i Ogonek Small	P1-415 (0xBF)	0x1EBF
416	Į	i Ogonek Capital	P1-416 (0xC0)	0x1EC0
417	ij	ij Ligature Small	P1-417 (0xC1)	0x1EC1
418	IJ	IJ Ligature Capital	P1-418 (0xC2)	0x1EC2
419	ĵ	j Circumflex Small	P1-419 (0xC3)	0x1EC3
420	Ĵ	j Circumflex Capital	P1-420 (0xC4)	0x1EC4
421	ķ	k Cedilla Small	P1-421 (0xC5)	0x1EC5
422	Ķ	k Cedilla Capital	P1-422 (0xC6)	0x1EC6
423	ĸ	k Greenlandic Small	P1-423 (0xC7)	0x1EC7
424	ļ	l Cedilla Small	P1-424 (0xC8)	0x1EC8
425	Ļ	l Cedilla Capital	P1-425 (0xC9)	0x1ec9
426	ŀ	l Middle Dot Small	P1-426 (0xCA)	0x1ECA
427	Ŀ	l Middle Dot Capital	P1-427 (0xCB)	0x1ECB
428	ņ	n Cedilla Small	P1-428 (0xCC)	0x1ECC
429	Ņ	n Cedilla Capital	P1-429 (0xCD)	0x1ECD
430	ŋ	n Eng Lapp Small	P1-430 (0xCE)	0x1ECE
431	Ŋ	n Eng Lapp Capital	P1-431 (0xCF)	0x1ECF
432	ō	o Macron Small	P1-432 (0xD0)	0x1ED0
433	Ō	o Macron Capital	P1-433 (0xD1)	0x1ED1
434	œ	oe Ligature Small	P1-434 (0xD2)	0x1ED2
435	Œ	oe Ligature Capital	P1-435 (0xD3)	0x1ED3
436	ŗ	r Cedilla Small	P1-436 (0xD4)	0x1ED4
437	Ŗ	r Cedilla Capital	P1-437 (0xD5)	0x1ED5
438	ŝ	s Circumflex Small	P1-438 (0xD6)	0x1ED6
439	Ŝ	s Circumflex Capital	P1-439 (0xD7)	0x1ED7
440	ŧ	t Stroke Small	P1-440 (0xD8)	0x1ED8

Figure 3-31 (Part 5 of 7). Code Page P1

Position Number	Character		Code Page/ Code Point	Internal Code
441	Ŧ	t Stroke Capital	P1-441 (0xD9)	0x1ED9
442	ũ	u Tilde Small	P1-442 (0xDA)	0x1EDA
443	Ũ	u Tilde Capital	P1-443 (0xDB)	0x1EDB
444	ŭ	u Breve Small	P1-444 (0xDC)	0x1EDC
445	Ŭ	u Breve Capital	P1-445 (0xDD)	0x1EDD
446	ū	u Macron Small	P1-446 (0xDE)	0x1EDE
447	Ū	u Macron Capital	P1-447 (0xDF)	0x1EDF
448	ų	u Ogonek Small	P1-448 (0xE0)	0x1EE0
449	Ų	u Ogonek Capital	P1-449 (0xE1)	0x1EE1
450	ŵ	w Circumflex Small	P1-450 (0xE2)	0x1EE2
451	Ŵ	w Circumflex Capital	P1-451 (0xE3)	0x1EE3
452	ŷ	y Circumflex Small	P1-452 (0xE4)	0x1EE4
453	Ŷ	y Circumflex Capital	P1-453 (0xE5)	0x1EE5
454	Ÿ	y Umlaut Capital	P1-454 (0xE6)	0x1EE6
455	©	Copyright Symbol	P1-455 (0xE7)	0x1EE7
456	¹	Superscript One	P1-456 (0xE8)	0x1EE8
457	™	Trademark Symbol	P1-457 (0xE9)	0x1EE9
458	⅛	One Eighth	P1-458 (0xEA)	0x1EEA
459	⅜	Three Eights	P1-459 (0xEB)	0x1EEB
460	⅝	Five Eighths	P1-460 (0xEC)	0x1EEC
461	⅞	Seven Eighths	P1-461 (0xED)	0x1EED
462	×	Multiplication Sign	P1-462 (0xEE)	0x1EEE
463	’	Right Single Quote	P1-463 (0xEF)	0x1EEF
464	“	Left Double Quote	P1-464 (0xF0)	0x1EF0
465	”	Right Double Quote	P1-465 (0xF1)	0x1EF1
466	=	Equal Sign Superscript	P1-466 (0xF2)	0x1EF2
467	-	Minus Sign Superscript	P1-467 (0xF3)	0x1EF3
468	+	Plus Sign Superscript	P1-468 (0xF4)	0x1EF4
469	∞	Infinity symbol Superscript	P1-469 (0xF5)	0x1EF5
470	π	Pi Symbol Superscript	P1-470 (0xF6)	0x1EF6
471	Δ	Delta Symbol Superscript	P1-471 (0xF7)	0x1EF7
472	→	Right Arrow Superscript	P1-472 (0xF8)	0x1EF8
473	/	Slash Superscript	P1-473 (0xF9)	0x1EF9
474	†	Dagger	P1-474 (0xFA)	0x1EFA
475	<	Left Angle Superscript	P1-475 (0xFB)	0x1EFB
476	>	Right Angle Superscript	P1-476 (0xFC)	0x1EFC
477	℞	Prescription Symbol	P1-477 (0xFD)	0x1EFD

Figure 3-31 (Part 6 of 7). Code Page P1

Position Number	Character	Code Page/ Code Point	Internal Code
478	∉ 'Is Not An Element' Symbol	P1-478 (0xFE)	0x1EFE
479	∴ 'Therefore' Symbol	P1-479 (0xFF)	0x1EFF

Figure 3-31 (Part 7 of 7). Code Page P1

Position Number	Character		Code Page/ Code Point	Internal Code
480	↗	Increase	P2-480 (0x20)	0x1DA0
481	↘	Decrease	P2-481 (0x21)	0x1DA1
482	‡	Double Dagger	P2-482 (0x22)	0x1DA2
483	≠	Not Equal Symbol	P2-483 (0x23)	0x1DA3
484	∨	OR Symbol	P2-484 (0x24)	0x1DA4
485	∧	AND Symbol	P2-485 (0x25)	0x1DA5
486	‖	Parallel	P2-486 (0x26)	0x1DA6
487	∠	Angle Symbol	P2-487 (0x27)	0x1DA7
488	<	Left Angle Bracket	P2-488 (0x28)	0x1DA8
489	>	Right Angle Bracket	P2-489 (0x29)	0x1DA9
490	∓	Minus Or Plus Sign	P2-490 (0x2A)	0x1DAA
491	□	Lozenge	P2-491 (0x2B)	0x1DAB
492	′	Minutes Symbol	P2-492 (0x2C)	0x1DAC
493	∫	Integral Symbol	P2-493 (0x2D)	0x1DAD
494	∪	Union	P2-494 (0x2E)	0x1DAE
495	⊂	'Is Included In' Symbol	P2-495 (0x2F)	0x1DAF
496	⊃	'Includes' Symbol	P2-496 (0x30)	0x1DB0
497	⊕	Circle Plus, Closed Sum	P2-497 (0x31)	0x1DB1
498	∟	Right Angle Symbol	P2-498 (0x32)	0x1DB2
499	⊗	Circle Multiply	P2-499 (0x33)	0x1DB3
500	"	Seconds Symbol	P2-500 (0x34)	0x1DB4
501	‗	Double Overline	P2-501 (0x35)	0x1DB5
502	ψ	Psi Small	P2-502 (0x36)	0x1DB6
503	ε	Epsilon Small	P2-503 (0x37)	0x1DB7
504	λ	Lambda Small	P2-504 (0x38)	0x1DB8
505	η	Eta Small	P2-505 (0x39)	0x1DB9
506	ι	Iota Small	P2-506 (0x3A)	0x1DBA
507	⎛	Upper Left Parenthesis Section	P2-507 (0x3B)	0x1DBB
508	⎝	Lower Left Parenthesis Section	P2-508 (0x3C)	0x1DBC
509	‰	Permille Symbol	P2-509 (0x3D)	0x1DBD
510	θ	Theta Small	P2-510 (0x3E)	0x1DBE
511	κ	Kappa Small	P2-511 (0x3F)	0x1DBF
512	ω	Omega Small	P2-512 (0x40)	0x1DC0
513	ν	Nu Small	P2-513 (0x41)	0x1DC1
514	ο	Omicron Small	P2-514 (0x42)	0x1DC2
515	ρ	Rho Small	P2-515 (0x43)	0x1DC3
516	γ	Gamma Small	P2-516 (0x44)	0x1DC4

Figure 3-32 (Part 1 of 4). Code Page P2

Position Number	Character		Code Page/ Code Point	Internal Code
517	ϑ	Theta Small	P2-517 (0x45)	0x1DC5
518	⎛	Upper Right Parenthesis Section	P2-518 (0x46)	0x1DC6
519	⎝	Lower Right Parenthesis Section	P2-519 (0x47)	0x1DC7
520	≅	'Congruent To' Symbol	P2-520 (0x48)	0x1DC8
521	ξ	Xi Small	P2-521 (0x49)	0x1DC9
522	χ	Chi Small	P2-522 (0x4A)	0x1DCA
523	υ	Upsilon Small	P2-523 (0x4B)	0x1DCB
524	ζ	Zeta Small	P2-524 (0x4C)	0x1DCC
525	⎰	Lower Right/Upper Left Brace Section	P2-525 (0x4D)	0x1DCD
526	⎱	Upper Right/Lower Left Brace Section	P2-526 (0x4E)	0x1DCE
527	$_0$	Zero Subscript	P2-527 (0x4F)	0x1DCF
528	$_1$	One Subscript	P2-528 (0x50)	0x1DD0
529	$_2$	Two Subscript	P2-529 (0x51)	0x1DD1
530	$_3$	Three Subscript	P2-530 (0x52)	0x1DD2
531	$_4$	Four Subscript	P2-531 (0x53)	0x1DD3
532	$_5$	Five Subscript	P2-532 (0x54)	0x1DD4
533	$_6$	Six Subscript	P2-533 (0x55)	0x1DD5
534	$_7$	Seven Subscript	P2-534 (0x56)	0x1DD6
535	$_8$	Eight Subscript	P2-535 (0x57)	0x1DD7
536	$_9$	Nine Subscript	P2-536 (0x58)	0x1DD8
537	⊥	Perpendicular	P2-536 (0x59)	0x1DD9
538	⏀	Total Symbol	P2-537 (0x5A)	0x1DDA
539	Ψ	Psi Capital	P2-538 (0x5B)	0x1DDB
540	Π	Pi Capital	P2-539 (0x5C)	0x1DDC
541	Λ	Lambda Capital	P2-541 (0x5D)	0x1DDD
542	⏚	Bottle Symbol	P2-542 (0x5E)	0x1DDE
543	␢	Substitute Blank	P2-543 (0x5F)	0x1DDF
544	∂	Partial Differential Symbol	P2-544 (0x60)	0x1DE0
545	⌁	Sine Symbol	P2-545 (0x61)	0x1DE1
546	□	Open Square	P2-546 (0x62)	0x1DE2
547	■	Solid Square	P2-547 (0x63)	0x1DE3
548	▨	Slash Square	P2-548 (0x64)	0x1DE4
549	⎲	Upper Summation Section	P2-549 (0x65)	0x1DE5
550	⎳	Lower Summation Section	P2-550 (0x66)	0x1DE6
551	Ξ	Xi Capital	P2-551 (0x67)	0x1DE7
552	$\propto$	'Proportional To' Symbol	P2-552 (0x68)	0x1DE8
553	Δ	Delta Capital	P2-553 (0x69)	0x1DE9

Figure 3-32 (Part 2 of 4). Code Page P2

Position Number	Character		Code Page/ Code Point	Internal Code
554	ϒ	Upsilon Capital	P2-554 (0x6A)	0x1DEA
555	≃	'Approximately Equal To' Symbol	P2-555 (0x6B)	0x1DEB
556	~	Cycle Symbol, 'Equivalent To' Symbol	P2-556 (0x6C)	0x1DEC
557	⁰	Zero Superscript	P2-557 (0x6D)	0x1DED
558	⁴	Four Superscript	P2-558 (0x6E)	0x1DEE
559	⁵	Five Superscript	P2-559 (0x6F)	0x1DEF
560	⁶	Six Superscript	P2-560 (0x70)	0x1DF0
561	⁷	Seven Superscript	P2-561 (0x71)	0x1DF1
562	⁸	Eight Superscript	P2-562 (0x72)	0x1DF2
563	⁹	Nine Superscript	P2-563 (0x73)	0x1DF3
564	Ø	Zero Slash	P2-564 (0x74)	0x1DF4
565	₧	Paseta Sign	P2-565 (0x75)	0x1DF5
566	⌐	Flipped Logical Not	P2-566 (0x76)	0x1DF6
567	╡	Right Side Middle - Double Horizontal	P2-567 (0x77)	0x1DF7
568	╢	Right Side Middle - Double Vertical	P2-568 (0x78)	0x1DF8
569	╖	Upper Right Corner Box - Double Vertical	P2-569 (0x79)	0x1DF9
570	╕	Upper Right Corner Box - Double Hor.	P2-570 (0x7A)	0x1DFA
571	╜	Lower Right Corner Box - Double Vertical	P2-571 (0x7B)	0x1DFB
572	╛	Lower Right Corner Box - Double Hor.	P2-572 (0x7C)	0x1DFC
573	╞	Left Side Middle - Double Horizontal	P2-573 (0x7D)	0x1DFD
574	╟	Left Side Middle - Double Vertical	P2-574 (0x7E)	0x1DFE
575	╧	Bottom Side Middle - Double Horizontal	P2-575 (0x7F)	0x1DFF
576	╨	Bottom Side Middle - Double Vertical	P2-576 (0x80)	0x1C80
577	╤	Top Side Middle - Double Horizontal	P2-577 (0x81)	0x1C81
578	╒	Lower Left Corner Box - Double Vertical	P2-578 (0x82)	0x1C82
579	╘	Lower Left Corner Box - Double Hor.	P2-579 (0x83)	0x1C83
580	╔	Upper Left Corner Box - Double Hor.	P2-580 (0x84)	0x1C84
581	╓	Upper Left Corner Box - Double Vertical	P2-581 (0x85)	0x1C85
582	╫	Intersection - Double Vertical	P2-582 (0x86)	0x1C86
583	╪	Intersection - Double Horizontal	P2-583 (0x87)	0x1C87
584	▌	Bright Character Cell - Left Half	P2-585 (0x88)	0x1C88
585	▐	Bright Character Cell - Right Half	P2-585 (0x89)	0x1C89
586	α	Alpha Small	P2-586 (0x8A)	0x1C8A
587	β	Beta Small	P2-587 (0x8B)	0x1C8B
588	Γ	Gamma Capital	P2-588 (0x8C)	0x1C8C
589	π	Pi Small	P2-589 (0x8D)	0x1C8D
590	Σ	Sigma Capital/Summation Sign	P2-590 (0x8E)	0x1C8E

Figure 3-32 (Part 3 of 4). Code Page P2

Position Number	Character		Code Page/ Code Point	Internal Code
591	σ	Sigma Small	P2-591 (0x8F)	0x1C8F
592	τ	Tau Small	P2-592 (0x90)	0x1C90
593	Φ	Phi Capital	P2-593 (0x91)	0x1C91
594	Θ	Theta Capital	P2-594 (0x92)	0x1C92
595	Ω	Omega Capital/Ohm Sign	P2-595 (0x93)	0x1C93
596	δ	Delta Small	P2-596 (0x94)	0x1C94
597	∞	Infinity	P2-597 (0x95)	0x1C95
598	φ	Phi Small	P2-598 (0x96)	0x1C96
599	∈	'Is An Element Of' Symbol	P2-599 (0x97)	0x1C97
600	∩	Intersection	P2-600 (0x98)	0x1C98
601	≡	Identity Symbol	P2-601 (0x99)	0x1C99
602	≥	'Greater Than or Equal To' Symbol	P2-602 (0x9A)	0x1C9A
603	≤	"Less Than or Equal To' Symbol	P2-603 (0x9B)	0x1C9B
604	⌠	Upper Intregal Section	P2-604 (0x9C)	0x1C9C
605	⌡	Lower Integral Section	P2-605 (0x9D)	0x1C9D
606	≈	Double Equivalent	P2-606 (0x9E)	0x1C9E
607	∙	Solid Overcircle	P2-607 (0x9F)	0x1C9F
608	√	Radical Symbol, Square Root	P2-608 (0xA0)	0x1CA0
609	⫟	Top Side Middle - Double Vertical	P2-609 (0xA1)	0x1CA1
610	ⁿ	Superscript n	P2-610 (0xA2)	0x1CA2
611		Numeric Space	P2-611 (0xA3)	0x1CA3
612	℄	Center Line	P2-612 (0xA4)	0x1CA4
613	⌴	Counter Bore	P2-613 (0xA5)	0x1CA5
614	⌵	Counter Sink	P2-614 (0xA6)	0x1CA6
615	↧	Depth	P2-615 (0xA7)	0x1CA7
616	⌀	Diameter	P2-616 (0xA8)	0x1CA8

Figure 3-32 (Part 4 of 4). Code Page P2

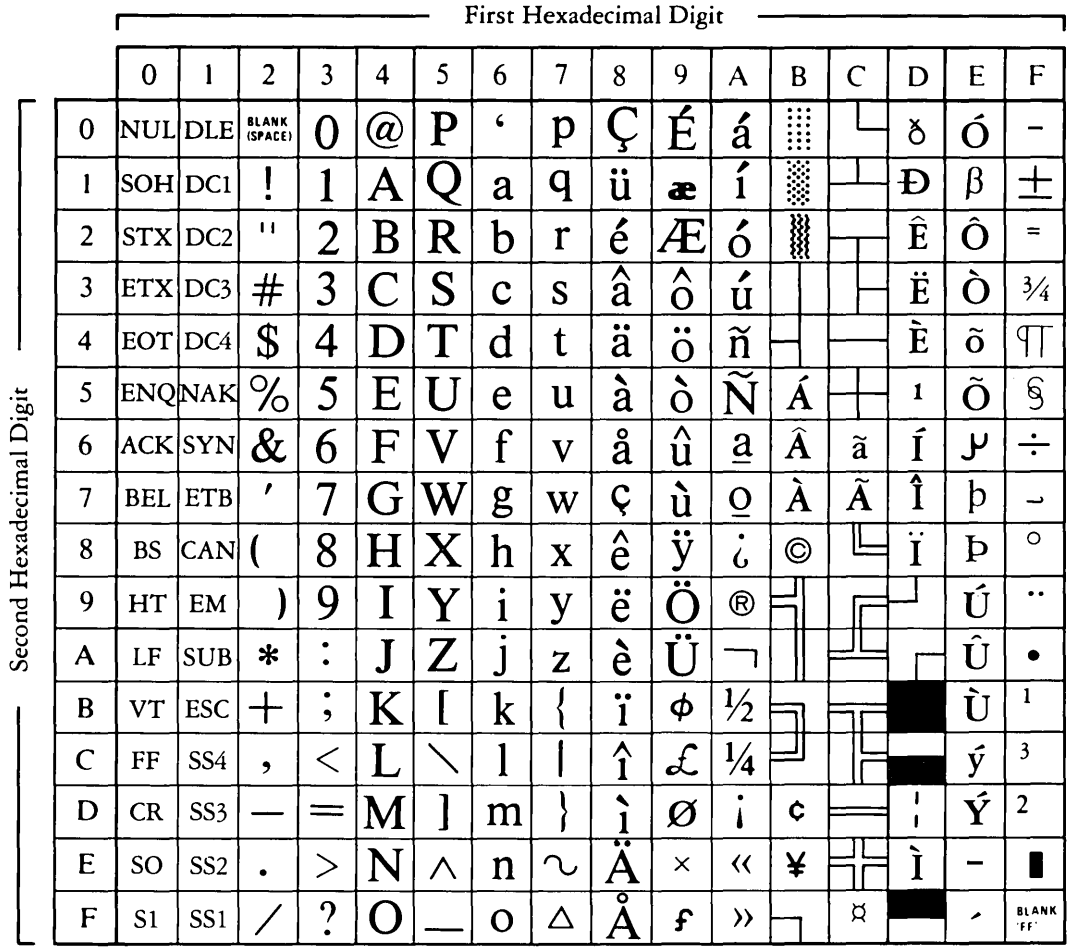


Figure 3-33. Code Page P0

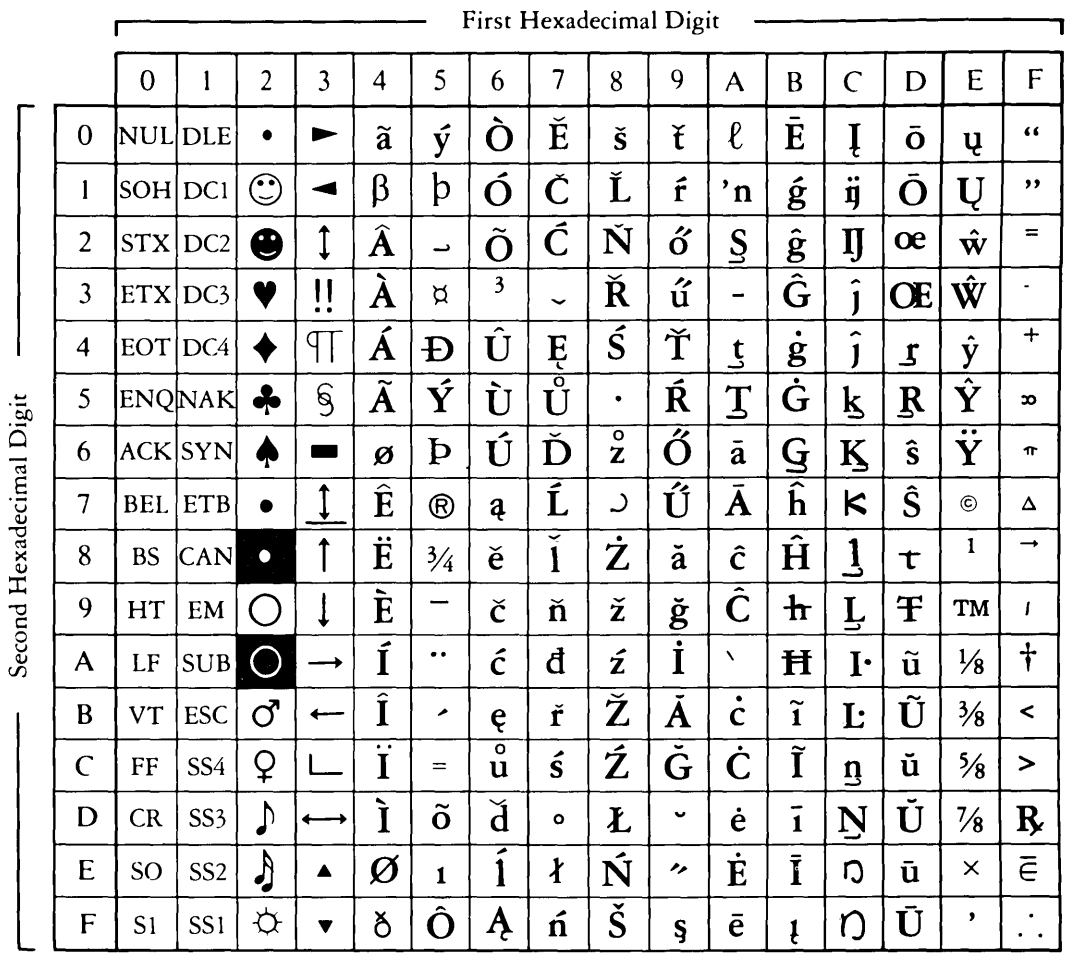


Figure 3-34. Code Page P1

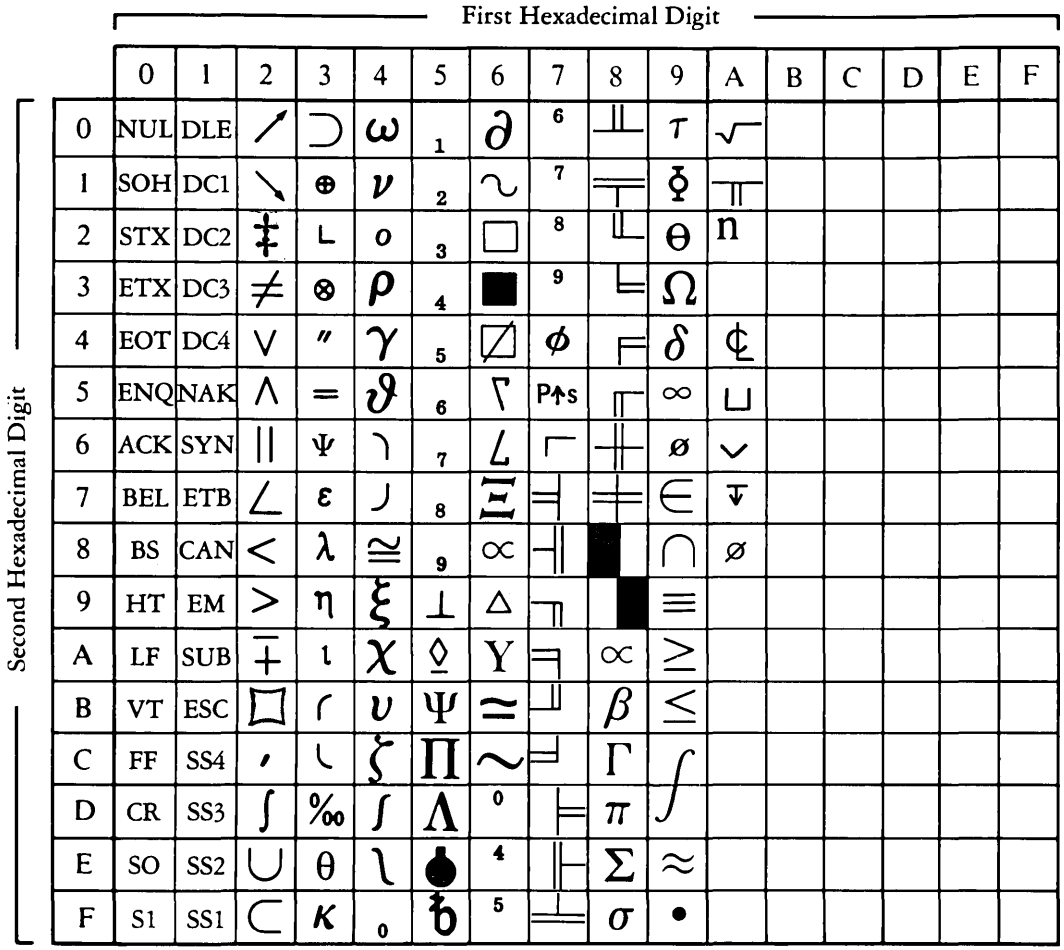


Figure 3-35. Code Page P2

Display Device Driver Considerations

A display device driver is somewhat more complex than a typical hardware device driver. Most hardware devices, such as a diskette drive or printer, have a device driver and perhaps a device manager that regulate the use of the device. A display device has the additional layer of software components described in "Virtual Machine Interface to the VTRM" on page 3-6 and "Virtual Machine Interface to the VTMP" on page 3-26. Still more software components exist between the VTMP and the hardware device driver. Two modules, the ***common device utilities*** and ***device-specific modules*** (both shown in Figure 3-36), provide additional support for display device implementation. The common device utilities are available to the VTMP to perform such tasks as querying display or locator devices, setting parameters for display and keyboard functions, and so on. The device-specific modules (DSM) are a set of callable procedures that, along with the device driver, manipulate and control the contents of the physical display device. In order to provide support for a new physical display, therefore, a developer must integrate a DSM and its accompanying display device driver into the VRM according to the specifications described here.

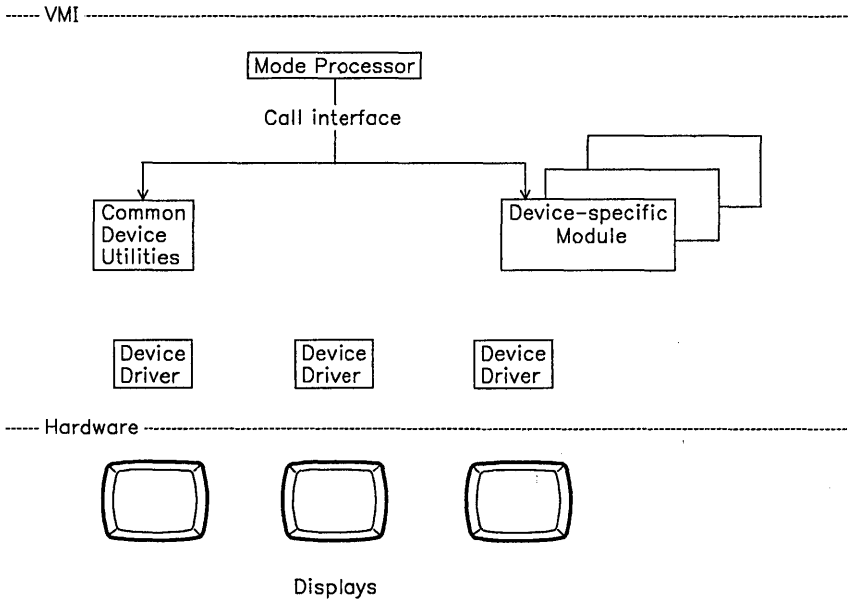


Figure 3-36. Display Subsystem Components

Device-specific Module Characteristics

A device-specific module can be in one of three modes and in either active or inactive state within a mode. Valid modes include:

- Monitored mode (MOM)

  In MOM mode the DSM relinquishes all control of the physical display to the calling process. The DSM detaches from its device driver and sets its internal state to refrain from modifying the display in any way. In MOM mode, an application uses the VTMP to take direct control of the display without interference from the DSM. All DSMs support a MOM mode.

- Keyboard send/receive mode (KSR)

  In KSR mode, all DSMs maintain a presentation space buffer that presents an application programming interface to all displays as if the display had a character buffer. This presentation space buffer is a two-dimensional alphanumeric character array that maps characters onto the display when the DSM is active. In KSR mode, the calling process can freely modify the presentation space whether the DSM is active or inactive. All DSMs support a KSR mode.

- All-points-addressable mode (APA)

  DSMs that support display devices with all-points-addressable capability can optionally implement an APA mode. In APA mode, a virtual bit map (APA buffer) allocated and managed by the caller can be copied to the display by the DSM. The caller has the option of specifying a sub-rectangle of the APA buffer (known as the ***external raster array***) to be copied to the screen. This is useful when only a portion of the virtual bit map has been modified and the caller wants to display it. Whether a DSM and its physical display can operate in APA mode is determined by the DDS of the display device driver.

The mode can be changed at any time with the Set Mode function ("Set Mode" on page 3-170) regardless of the DSM state.

The two states of a DSM are defined as follows:

- Active

  In active state, all DSM functions result in modifications to the display hardware. Only one instance of a DSM can be in active state at any one time.

- Inactive

  The inactive state allows multiple virtual terminals to be implemented because it allows DSM functions to be freely called without modifying the display hardware. In KSR mode, the application can function at will, and all output to the DSM modifies only the RAM-resident virtual buffers. In MOM mode, the application determines the functions to implement. In APA mode, the caller must not issue the Update External Raster Array function ("Update External Raster Array" on page 3-173) when inactive.

Device-specific module entry points

The following entry points are required for display device-specific modules added to the VRM. All these entry points must be present in the module, even if they perform no function. Because the VTMP calls each entry point as required (although the Initialize function is **always** called first), there is no order dependency for the entry points. In addition, there must be **no** main entry point defined.

Activate

Description: This function switches the DSM into active state and gives the module exclusive access to the physical display. The mode is set according to the last Set Mode function issued. Each mode has certain characteristics regarding hardware interrupts and presentation space buffers. Those characteristics are:

- Monitored mode

  To activate monitored mode on a device that uses hardware interrupts, the device driver is detached from the previously active instance of the DSM.

- KSR mode

  If the current state of the adapter is not KSR mode, the device is initialized to KSR mode. For every instance of KSR mode, the Activate function causes the current contents of the presentation space buffer to be drawn to the hardware refresh buffer.

  If the device uses hardware interrupts, the calling process is attached to the device driver for interrupt-handling purposes, then is detached. This keeps the VRM from calling the interrupt-handling entry point more often than necessary.

  If the device has a loadable color table, the DSM's KSR mode color table is loaded into the adapter's real color table.

- APA mode

  If the current state of the adapter is not APA mode, the device is initialized to APA mode. To complete the switch to APA mode, an Update External Raster Array function must be called next (see "Update External Raster Array" on page 3-173).

  If the device uses hardware interrupts, the calling process is attached to the device driver for interrupt-handling purposes, then is detached. This keeps the VRM from calling the interrupt-handling entry point more often than necessary.

  If the device has a loadable color table, the DSM's APA mode color table is loaded into the adapter's real color table.

Calling Sequence:

```
cvttact ( )
```

This function passes no parameters.

Programming Notes:

Only one instance of the device driver can be in the activated state at any one time. Calling the Activate entry point when the device driver is already activated will yield unpredictable results for subsequent operations.

Note that the Initialize function is called before the first call to Activate.

This function can be called from any of the device-specific module modes.

IBM recommends the following approach to coding this entry point:

1. Mark this instance of the device-specific module as active.
2. If the instance-unique mode of the DSM is monitored mode, then:
   a. Set the device and real screen table modes to MOM for this instance of the DSM.
   b. If a process is attached to the device driver, detach the process and set the `rscr_path[rscr_id-1]` field of the real screen table to zero.
3. If the instance-unique mode of the DSM is KSR mode, then:
   a. Set the device and real screen table modes to KSR for this instance of the DSM.
   b. Ensure that a process is attached to the device driver (the `rscr_path[rscr_id-1]` field of the real screen table must not equal zero). If this field is zero (no process attached), attach a process to the device driver. If you are using interrupts, attach your own process to the driver. If you are not using interrupts, use the **vtrm_rc** process to the driver. The **vtrm_rc** ID can be obtained from the shared resource structure.
   c. Next, set the `rscr_path[rscr_id-1]` field of the real screen table to the path ID returned from the attach.
   d. Copy the contents of the RAM-resident presentation space to the physical refresh buffer, doing any required device-specific raster conversion.
   e. Draw or enable the cursor, returning it to the correct shape and position.
   f. Clear any unused areas at the right edge or bottom of the presentation space.
   g. Set the device's color table to the appropriate virtual color table for the current mode.
4. If the instance-unique mode of the DSM is APA mode, then:
   a. Set the device and real screen table modes to APA for this instance of the DSM.
   b. Ensure that a process is attached to the device driver (the `rscr_path[rscr_id-1]` field of the real screen table must not equal zero). If this field is zero (no process attached), attach a process to the device driver. If you are using interrupts, attach your own process to the driver. If you are not using interrupts, use the **vtrm_rc** process to the driver. The **vtrm_rc** ID can be obtained from the shared resource structure.
   c. Next, set the `rscr_path[rscr_id-1]` field of the real screen table to the path ID returned from the attach.

Clear Rectangle

Description: This function allows you to specify a rectangular area in the presentation space and store into each character position in that area any combination of the following attributes:

- Foreground color (one of 16 colors can be specified)
- Background color (one of 16 colors can be specified)
- Font (one of eight fonts can be specified)
- Underscore
- Reverse video
- Blink
- Bright
- Nondisplay (blank).

If the physical display does not use a hardware cursor, the cursor can be optionally displayed when this function returns to the caller. If the display has a hardware cursor, the cursor is always displayed on return to the caller.

Calling Sequence:

```
cvttclr (screen_pos, attr, cursor_pos, show_cursor)
   struct vtt_box_rc_parms *screen_pos;
   unsigned int attr;
   struct vtt_cursor *cursor_pos;
   unsigned int show_cursor;
```

The parameters to Clear Rectangle are defined as follows:

- `screen_pos`

  This parameter specifies the coordinates of the upper left and lower right corners of the rectangular area to be cleared.

- `attr`

  This indicates the attributes of each character in the specified rectangular area.

- `cursor_pos`

  This indicates the x and y position to which the cursor is moved if `show_cursor` equals true.

- `show_cursor`

  This parameter indicates whether the cursor is redrawn at the end of the operation. This control allows the VTMP to determine the amount of APA display processing used to draw and redraw cursors. A 1 in this parameter indicates show cursor; a 0 indicates do not show cursor.

For displays that use a hardware sprite cursor (as the PC Monochrome A/N display does), this parameter indicates whether to move the cursor to the location specified by the input parameters. When `cursor_show` equals zero, the cursor is not moved; when it equals one, the cursor is moved.

Programming Notes:

If an invalid rectangular area is specified (such as coordinates outside the presentation space, upper left and lower right coordinates switched, and so on), the results of this function are unpredictable.

This function is valid only in KSR mode.

Copy Full Lines

Description: This function copies some or all lines in the presentation space to a new location either before or after the specified lines. The source and destination points must be within the presentation space and any information that lies beyond the lower right hand corner of the presentation space is truncated.

This function is used with the clear rectangular text area to allow the insertion of new lines into the presentation space.

The cursor can be optionally removed or remain displayed on the screen.

Calling Sequence:

```
cvttcfl (source_row, dest_row, length, cp, vtt_cursor_show)
   int source_row;
   int dest_row;
   int length;
   struct vtt_cursor_pos *cp;
   int vtt_cursor_show;
```

The parameters to the Copy Full Lines function are defined as follows:

- `source_row`

  This parameter specifies the first row number to copy.

- `dest_row`

  This parameter specifies the row number to which the copy operation will copy the first line of the specified source.

- `length`

  This indicates the number of lines to copy.

- `cp`

  This parameter stands for 'code point' and consists of the `cp` structure defined for the Draw Text function. For the Copy Full Lines function, however, all fields are reserved except for `vtt_cursor`. This parameter represents a structure that indicates the position to which the cursor is moved if `cursor_show` equals true.

The vtt_cursor structure is defined as follows:

```
struct vtt_cursor
{
   int cursor_x;
   int cursor_y;
} ;
```

- vtt_cursor_show

  This parameter indicates whether the cursor is redrawn at the end of the operation. This control allows the VTMP to determine the amount of APA display processing used to draw and redraw cursors. A 1 in this parameter indicates show cursor; a 0 indicates do not show cursor.

  For displays that use a hardware sprite cursor (as the PC Monochrome A/N display does), this parameter indicates whether to move the cursor to the location specified by the input parameters. When cursor_show equals zero, the cursor is not moved; when it equals one, the cursor is moved.

Programming Notes:

This function is valid only in KSR mode.

Copy Line Segment

Description: This function copies the partial or entire contents of a specified line in the presentation space and truncates any information that exceeds the absolute left or right margin of the line.

The contents of lines preceding the specified line are not affected, but succeeding lines can be copied in the same way if the number of lines specified is greater than one.

This function is used with the clear rectangular text area to obtain the repetitive intraline move function.

If the starting point happens to be the first column of the first row, and the number of lines requested is equal to the number of rows in the presentation space, then the entire screen is effectively scrolled right. Otherwise, a partial screen scroll is achieved.

The cursor can optionally be removed or remain displayed on the screen.

Calling Sequence:

```
cvttcpl (rc, cp, cursor_show)
   struct vtt_rc_parms *rc;
   struct vtt_cursor_pos *cp;
   int cursor_show;
;
```

The parameters to Copy Line Segment are defined as follows:

- `rc`

  This parameter stands for 'row/column' and consists of the following structure:

```
struct vtt_rc_parms
{
   short string_length;
   short string_index;
   int start_row;
   int start_column;
   int dest_row;
   int dest_column;
};
```

The fields in the preceding structure are defined as follows:

- `string_length`

  This length value specifies the number of character/attribute pairs to be copied in each line.

- `string_index`

  This field is reserved.

- `start_row`

  This indicates the first row number of the presentation space/buffer to copy to.

- `start_column`

  This indicates the starting position, within each line, of the column from which to copy.

- `dest_row`

  This indicates the last row number of the presentation space/buffer to which to copy.

- `dest_column`

  This indicates the starting position, within each line, of the destination to copy to. If the destination column number is greater than the starting column, the specified line segment is copied to the right. Otherwise, the line segment is copied to the left.

- `cp`

  This parameter stands for 'code point' and consists of the `cp` structure defined for the Draw Text function. For the Copy Line Segment function, however, all fields are reserved except for `vtt_cursor`. This parameter represents a structure that indicates the position to which the cursor is moved if `cursor_show` equals true. The `vtt_cursor` structure is defined as follows:

```
struct vtt_cursor
{
   int cursor_x;
   int cursor_y;
} ;
```

- `cursor_show`

  This parameter indicates whether the cursor is redrawn at the end of the operation. This control allows the VTMP to determine the amount of APA display

processing used to draw and redraw cursors. A 1 in this parameter indicates show cursor; a 0 indicates do not show cursor.

For displays that use a hardware sprite cursor (as the PC Monochrome A/N display does), this parameter indicates whether to move the cursor to the location specified by the input parameters. When `cursor_show` equals zero, the cursor is not moved; when it equals one, the cursor is moved.

Programming Notes:

This function is valid only in KSR mode.

Deactivate

Description: This function switches the DSM into the inactive state. The display device driver copies the contents of the hardware refresh buffer to the RAM-resident presentation buffer, if necessary. Subsequent move, draw, or cursor movement commands operate on the RAM-resident version of the presentation space instead of the hardware.

Calling Sequence:

```
cvttdact ( )
```

This function passes no parameters.

Programming Notes:

This function can be called from any of the DSM modes.

IBM recommends the following approach to coding this entry point:

1. Mark the state of this instance of the DSM to inactive.
2. If your device support strategy calls for the presentation space to be maintained in the device refresh buffer, then copy the RAM-resident version of the buffer back to the presentation space.

Define Cursor

Description: This function allows the cursor to be defined in one of six predefined shapes. In addition to the predefined shapes, the cursor can also be defined as a character box in which all the pixels are highlighted.

In any case, the width of the cursor equals the width of the character box.

The predefined cursor shapes for the PC Monochrome A/N display and adapter are defined as follows:

- Selection 0 – no cursor displayed (invisible)
- Selection 1 – single underscore (default)
- Selection 2 – double underscore
- Selection 3 – half blob (lower half)
- Selection 4 – mid-character double line
- Selection 5 – full blob

Selectors 6 through 254 are reserved. Selector 255 indicates that the top and bottom parameters to Define Cursor specify the top and bottom scan lines of the character box. The top scan line in a box is scan line 0; the bottom scan line is n, where n equals the height of the character box minus one. If the top value is greater than the bottom value, all scan lines in the character box are visible.

The cursor can also be repositioned with this function.

Calling Sequence:

```
cvttdefc (selector, cursor_pos, show_cursor, top, bot)
   unsigned char selector;
   struct vtt_cursor *cursor_pos;
   int show_cursor;
   int top;
   int bot;
```

The parameters to Define Cursor are defined as follows:

- `selector`

  This is the value of the cursor shape (from the preceding list).

- `cursor_pos`

  This indicates the x and y position to which the cursor is moved if `show_cursor` equals true.

- `show_cursor`

  This parameter indicates whether the cursor is redrawn at the end of the operation. This control allows the VTMP to determine the amount of APA display

processing used to draw and redraw cursors. A 1 in this parameter indicates show cursor; a 0 indicates do not show cursor.

For displays that use a hardware sprite cursor (as the PC Monochrome A/N display does), this parameter indicates whether to move the cursor to the location specified by the input parameters. When `cursor_show` equals zero, the cursor is not moved; when it equals one, the cursor is moved.

- `top`

  This parameter is valid only if the selector value is 255. In this case, `top` indicates the first line in the character box to make visible.

- `bot`

  This parameter is valid only if the selector value is 255. In this case, `bot` indicates the last line in the character box to make visible.

Programming Notes:

The cursor can be made invisible by specifying a selector value of zero, regardless of the `show_cursor` value.

If an invalid cursor position is specified, the results are unpredictable.

This function is valid only in KSR mode.

Draw Text

Description: This function draws a string of code-based qualified ASCII characters into the refresh buffer and presentation space buffer of the display.

Each supplied ASCII character is drawn into the buffer/presentation space at the specified row and column. After the character is drawn, the cursor can be redrawn at the specified new location.

Each character drawn is mapped to the appropriate font range by two mechanisms. They are:

- Each character is logically ANDED by the value supplied in the code point mask parameter. This parameter should contain either 0xFF or 0x7B, which effectively wraps the code point in a 7-bit or 8-bit range.
- The code point base parameter is also added to each supplied character to find the correct symbol in the font.

Each character is drawn with a device-specific attribute derived from the canonical attribute specification in the `vtt_attr` parameter of the `vtt_cp_parms` structure.

The cursor can be moved or redrawn by the Draw Text function if the `vtt_show` parameter bit is set to one. In this case, a cursor of the type defined by the Define Cursor function is moved to or redrawn at the location specified in the `cursor_x` and `cursor_y` parameters. If the show bit is set to zero, the cursor may or may not be invisible after the Draw Text operation, as this is a device-dependent characteristic.

Calling Sequence:

```
cvttrep (ascii_string, rc, cp, vtt_show)
   char *ascii_string;
   struct vtt_rc_parms *rc;
   struct vtt_cp_parms *cp;
   int vtt_show;
```

The parameters to Draw Text are defined as follows:

- `ascii_string`

  This parameter is a reference to an adjustable array of characters to be drawn on the display. The length of this string must be greater than or equal to the length parameter field in the `vtt_rc_parms` structure. This string is not null terminated.

- rc

  This parameter stands for 'row/column' and consists of the following structure:

```
struct vtt_rc_parms
{
   short string_length;
   short string_index;
   int start_row;
   int start_column;
   int dest_row;
   int dest_column;
};
```

  The fields in the preceding structure are defined as follows:

  - string_length

    This length value specifies the number of characters in the ASCII string to draw. Note that the Draw Text function performs no end-of-line check, so unpredictable results can occur if the string length and start column values cause the characters to be written off the end of a line.

  - string_index

    This indicates the first character in the ASCII_STRING to display.

  - start_row

    This indicates the row number of the presentation space/buffer of the first character to be drawn. This parameter has a unity origin.

  - start_column

    This indicates the column number of the presentation space/buffer of the first character to be drawn. This parameter has a unity origin.

  - dest_row

    This indicates the destination row number (zero based) for move operations.

  - dest_column

    This indicates the destination column number (zero based) for move operations.

- cp

  This parameter stands for 'code point' and consists of the following structure:

```
struct vtt_cp_parms
{
 unsigned int cp_mask;
 int cp_base;
 struct vtt_cursor cursor;
 union
 {
   struct
   {
      short rsvd1_1;
      unsigned short value;
  } vtt_attr_val;
   struct attributes
   {
     short rsvd_2;
     unsigned fg_color : 4;
     unsigned bg_color : 4;
     unsigned font_select : 3;
     unsigned no_disp : 1;
     unsigned bright : 1;
     unsigned blink : 1;
     unsigned rev_video : 1;
     unsigned underscore : 1;
   } vtt_attr;
  } attrib;
} ;
```

  The fields in the preceding structure are defined as follows:

  - cp_mask

    This parameter indicates the bit (8) value that is logically ANDED with each ASCII character before font translation. Valid values for this parameter are 0xFF and 0x7F.

  - cp_base

    This parameter contains a value that allows each ASCII character to be translated to a final font set larger than 256 code points. The ASCII character

is masked by `vtt_cp_parms.cp_mask`. The masked ASCII character is added to the value of `vtt_cp_parms.cp_base` to determine the actual character (in a 10-bit selection) to display.

- `vtt_cursor`

  This parameter represents a structure that indicates the values to use as the new cursor column and row positions (if the cursor is to be displayed at the end of the Draw Text operation). In the following structure, the x value is for the column and the y value is for the row.

```
struct vtt_cursor
{
   int cursor_x;
   int cursor_y;
} ;
```

- `vtt_attr_val`

  This parameter represents a structure that specifies the attributes to use when drawing a character string on the display. The structure is defined as follows:

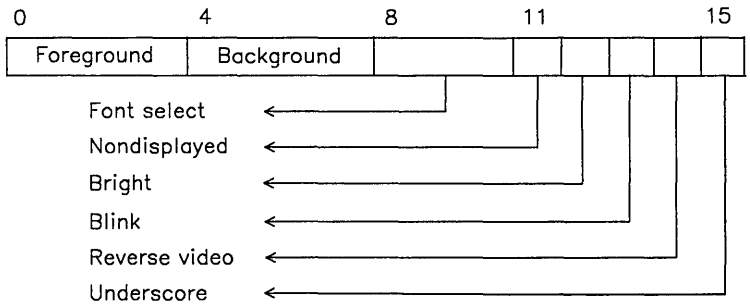


Figure 3-37. Attribute Structure for Draw Text Function

The fields in the attribute structure are defined as follows:

- Foreground – indicates the foreground color to be used when writing into the frame buffer. The default value is device-dependent. The PC monochrome adapter returns a default value of 2 (green).
- Background – indicates the background color to be used when writing into the frame buffer. The default value is device-dependent. The PC monochrome adapter returns a default value of 0 (black).
- Font select – selects the font to use when writing into the frame buffer. A value of 0 selects the first font ID supplied in the `vttinit` font ID array

as the active font, a value of 3 selects the fourth ID supplied in the array, and so on. The default value is 0.

- Nondisplayed – determines whether the character is displayed. A value of 0 indicates character displayed; a value of 1 indicates character not displayed. The default value is 0.
- Bright – determines whether the character is displayed brightly (intensified). A value of 0 indicates normal intensity; a value of 1 indicates intensified display. The default value is 0.
- Blink – determines whether the character representation blinks. A value of 0 indicates non-blink; a value of 1 indicates blink the character on the display. The default value is 0.
- Reverse video – indicates whether the characters are displayed in reverse video. A value of 0 indicates normal image; a value of 1 indicates reverse video. The default value is 0.
- Underscore – indicates whether characters are displayed with an underscore. A value of 0 indicates no underscore; a value of 1 indicates display characters underscored. The default value is 0.

- `vtt_show`

  This parameter indicates whether the cursor is redrawn at the end of the Draw Text operation. This control allows the VTMP to determine the amount of APA display processing used to draw and redraw cursors. A 1 in this parameter indicates show cursor; a 0 indicates do not show cursor.

  Note that if the display device uses a hardware sprite cursor (like the PC Monochrome A/N display does), this function affects cursor movement, not visibility. For a sprite cursor, this parameter indicates whether the cursor is moved to the position specified by the input parameters.

Programming Notes:

This function is valid only in KSR mode and only for drawing a string to a single line of the display at a time. Length specifications that imply a wrap to the next line in the middle of a call are invalid.

Finis

Description: This function prevents interrupts from being issued to a virtual terminal and **must** be called when a virtual terminal closes. If a virtual terminal closes without calling this function, and the terminal receives a virtual interrupt, the system abends.

Calling Sequence:

```
cvttfins (ignore_lock)
   int *ignore_lock;
```

The parameter to Finis is defined as follows:

- `ignore_lock`

  This parameter is optional and should never be supplied by a VTMP or monitored mode process. Note that this entry point is also bound to the common device utilities modules (see Figure 3-36 on page 3-140), which can issue this call without the activate/deactivate protocol supported by the screen manager. If this parameter is not specified, or if it is specified and the high-order bit of the integer field is 0, locks on internal data structures are respected. Otherwise, the locks are ignored.

Programming Notes: For display adapters that generate interrupts serviced by the virtual terminal, this function ensures that the adapter is not reinitialized when the virtual terminal process terminates.

This function can be called when the DSM is in any of its three modes.

IBM recommends the following approach to coding this entry point:

1. Free any storage acquired (such as for the presentation space) with the VRM **_mfree** call.
2. Undefine the default color table definitions. See "Define Color Table" on page 3-185 for details on how to undefine the definitions.
3. If `ignore_lock` does not equal zero, decrement the real screen usage count (`rscr_usage[rscr_id-1]`) in the resource structure. Then if `rscr_usage` equals zero and `rscr_path` does not equal zero, detach the process that is attached to the device driver and return.
4. Lock the resource structure with the VRM call **_recv(rs_lock);**.
5. Decrement the real screen usage count `rscr_usage[rscr_id-1]` in the resource structure.
6. If `rscr_usage` equals zero and `rscr_path` does not equal zero, detach the process that is attached to the device driver.
7. Unlock the resource structure with the VRM call **_send(rs_lock);**.

Initialize

Description: This entry point initializes the internal state of the DSM and optionally allows an array of 8 font IDs to be passed to this procedure. If the array is passed, the following actions occur:

- Each ID in the list is validated, and any invalid IDs are flagged in the `invalid_font_ids` parameter.
- All characters in all fonts in the array are checked to ensure that they are the same size. Any font with a character box size that differs from the first font in the array is flagged in the `invalid_font_size` parameter.
- The state of the DSM remains unchanged if any font IDs are invalid or if all characters in the selected fonts are not the same size.

If a font array is not passed to this entry point, a default font is selected according to the following criteria:

1. An 80 character by 25 line presentation space with a plain font attribute is searched for first.
2. If no font conforming to number 1 is found, a font of any attribute that yields a presentation space of 80 characters by 25 lines is selected.
3. If no font conforming to numbers 1 or 2 is found, any valid font with a plain attribute is selected. If no such font is found, any valid font is selected.

If no valid font exists, the presentation space height parameter is set to -1 and the state of the DSM does not change.

If a font array is validated or a default font is found, the following actions occur:

- The presentation space is initialized with space characters and the canonical attribute of each character is set to zero.
- The physical display is cleared if the virtual terminal is active. A PC monochrome alphanumeric display, for example, is cleared to a black background.
- The KSR and PC default color palettes are allocated and initialized (see "Set Color Table" on page 3-169).
- The cursor is placed at the upper left corner of the presentation space and displayed on the physical display if the DSM is active.
- The default cursor shape (double underscore) is selected.

Calling Sequence:

```
cvttinit (vt_id, font_ids, ps_size, invalid_font_ids,
          invalid_font_size)
   short vt_id;
   short *font_ids;
   struct ps_s *ps_size;
   char *invalid_font_ids;
   char *invalid_font_size;
```

The parameters for this function are defined as follows:

- `vt_id`

  This value is determined by the initialization parameters passed to the VTMP by the VRM **_initp** routine.

- `font_ids`

  This array contains the IDs of the selected fonts. Possible values for this array are found in the real font table of the shared resource structure ("Shared Resource Structure" on page 3-176). If this parameter is not specified, a null pointer is passed and a default font is selected.

- `ps_size`

  This output parameter is set to the width and height (in characters) of the presentation space. The parameter is specified as follows:

  ```
  struct ps_s
  {
            unsigned int ps_width;
            unsigned int ps_height;
  } ;
  ```

  If `ps_width` equals -1, an invalid virtual terminal ID was specified. If `ps_height` equals -1, no valid default font could be found.

- `invalid_font_ids`

  This parameter indicates whether any of the font IDs passed in the array are invalid. A bit set equal to one indicates the corresponding font ID is invalid. For example, if this parameter has a binary value of 00000111, font IDs 6, 7, and 8 in the array are invalid. This parameter is optional and valid only when the `font_ids` parameter is specified.

- `invalid_font_size`

  This parameter indicates whether all specified fonts have characters of the same size. A bit set to one indicates the corresponding font has characters of a different size than the first font specified in the array. For example, if this parameter has a binary value of 01000000, the second font in the array has different sized characters than the first font in the array (and is therefore invalid). This parameter is optional and valid only when the `font_ids` parameter is specified.

Programming Notes:

- Note that at least one font for the display must result in a presentation space of 80 characters by 25 lines. If no such font is available, many AIX applications will be unusable.
- Each DSM must virtualize the set of eight active fonts, so that each instance of the module can have its own unique set of fonts.
- The DSM must be in character mode before this entry point is called.
- Presentation space size is calculated using the following formulas:

  PS height = the height of the physical screen in picture elements divided by the number of rows in a character box.

  PS width = the width of the physical screen in picture elements divided by the number of columns in a character box.

  All division is integer division.

IBM recommends the following approach to coding this entry point:

1. Validate the `vt_id` parameter by checking for a non-zero value in the `vt_iodn` field of the resource structure virtual terminal table. If this value is not valid, set the presentation space height to zero and the width to -1. Otherwise, set the height and width to zero.
2. If a font ID array was passed, validate that all eight entries are in the real font table and that each font class matches your device. If any of the fonts are invalid, set the appropriate bit in the invalid font IDs parameter and return. If all the IDs are valid, check that all the fonts are the same size. If any of the fonts are of different sizes, set the appropriate bits in the invalid size parameter and return. If all the fonts are valid, record them in the instance-unique static data area for future use. This is a good time to establish addressability to the fonts from the fields in the resource structure real font table.
3. If a font ID array was not passed, find a font in the real font table that results in an 80 column by 25 row presentation space. If more than one font meets this requirement, select one with a plain attribute. If a valid font is found, set the instance-unique static data for all eight fonts to this font ID. If no font meets this requirement, set the presentation space height to -1 and return. This is a good time to establish addressability to the fonts from the fields in the resource structure real font table.

4. Now, save the real screen index for the current display. This index can be found by using the `vt_id` parameter value as an index into the virtual terminal table of the resource structure.
5. Calculate the presentation space and save it in the instance-unique data field. Allocate enough storage from the heap to contain the presentation space by using the VRM call **_mallc**.
6. Initialize the presentation space pointer and length fields in the virtual terminal dump table of the shared resource structure. Call the VRM **_dumptbl** routine to initialize dumping this area.
7. If the dump table for your real screen index is not initialized, do it now and call **_dumptbl**. If your device is an APA device, do not attempt to dump the real refresh buffer. In this case, set both the pointer and length fields to zero.
8. Next, initialize all data relating to the dynamic presentation space. Set all attributes and colors to the default states and all characters to ASCII spaces.
9. Now initialize the virtual cursor.
10. If your DSM was active when the Initialize entry point was called, clear the screen, draw the default cursor, clear any unused regions at the right side or bottom of the presentation space, and set the physical device's virtual color table for the current mode.
11. Convey addressability to the default color tables to the VTMP by passing the address and length of these structures to the Define Color Table service routine (see "Define Color Table" on page 3-185).
12. Return.

Move Cursor

Description: This function moves the cursor to the specified row and column. If the virtual terminal is active and the cursor has been defined as a visible shape, the cursor will always be visible after the move.

Calling Sequence:

```
cvttmovc (cursor_pos)
   struct vtt_cursor *cursor_pos;
```

The parameter to Move Cursor is a structure that indicates the relative column (x value) and row (y value) to place the cursor. The structure is defined as follows:

```
struct vtt_cursor
{
   int cursor_x;
   int cursor_y;
} ;
```

Programming Notes:

If an invalid position is specified, the results of a Move Cursor function are unpredictable.

This function is valid only in KSR mode.

Read Screen Segment

Description: This function reads all or part of the each presentation space entry into a 4-byte value (a 2-byte display symbol and a 2-byte attribute).

The internal data representation, kept by the device driver for the presentation space, is probably different than the 4-byte format returned to the caller due to hardware dependencies and the 6K-byte presentation space limit for each virtual terminal. Each entry read from the presentation space must be mapped into the 4-byte format.

Calling Sequence:

```
cvttrds (fb_char, fb_c_size, fb_attr, fb_a_size, rc)
   unsigned short *fb_char;
   int fb_c_size;
   unsigned short *fb_attr;
   int fb_a_size;
   struct vtt_rc_parms *rc;
```

The parameters to Read Screen Segment are defined as follows:

- `fb_char`

  This parameter specifies the output array for the display symbols.

- `fb_c_size`

  This parameter specifies the size of the output array for the display symbols.

- `fb_attr`

  This parameter specifies the output array for the attributes.

- `fb_a_size`

  This parameter indicates the size of the output array for the attributes.

- rc

  This parameter stands for 'row/column' and consists of the following structure:

```
struct vtt_rc_parms
{
   short string_length;
   short string_index;
   int start_row;
   int start_column;
   int dest_row;
   int dest_column;
};
```

  The fields in the preceding structure are defined as follows:

  - string_length

    This parameter indicates the number of character/attribute pairs to read from the presentation space.

  - string_index

    This field is reserved.

  - start_row

    This indicates the row number from which to start the read.

  - start_column

    This indicates the column number from which to start the read.

  - dest_row

    This field is reserved. for move operations.

  - dest_column

    This field is reserved.

Programming Notes:

The caller must allocate space in which to return the attributes. This space must be large enough to contain all the information for the number of characters requested.

This function is valid only in KSR mode.

Scroll

Description: This function moves the contents of the presentation space up or down the specified number of lines. When scrolling up, the lines moved off the top of the screen are discarded. The specified number of lines added at the bottom of the screen are cleared to blanks with the attributes provided. When scrolling down, the lines moved off the bottom of the screen are discarded. The specified number of lines added at the top of the screen are cleared to blanks with the attributes provided.

Calling Sequence:

```
cvttscr (lines, attributes, cursor_pos, show)
   int lines;
   unsigned int attributes;
   struct vtt_cursor *cursor_pos;
   unsigned int show_cursor;
```

The parameters to the Scroll function are defined as follows:

- `lines`

  This parameter specifies the number of lines to scroll. A positive value moves the screen contents upward; a negative value moves the screen contents downward.

- `attributes`

  This specifies the attributes of the blanks to be inserted at the top or bottom of the screen.

- `cursor_pos`

  This indicates the x and y position to which the cursor is moved if `show_cursor` equals true.

- `show_cursor`

  This parameter indicates whether the cursor is redrawn at the end of the operation. This control allows the VTMP to determine the amount of APA display processing used to draw and redraw cursors. A 1 in this parameter indicates show cursor; a 0 indicates do not show cursor.

  For displays that use a hardware sprite cursor (as the PC Monochrome A/N display does), this parameter indicates whether to move the cursor to the location specified by the input parameters. When `cursor_show` equals zero, the cursor is not moved; when it equals one, the cursor is moved.

Programming Notes:

The Scroll function is valid only in KSR mode.

Set Color Table

Description: This function updates the DSM's KSR or APA color table when the module is in KSR or APA mode, respectively.

Color display adapters usually support a predetermined number of colors, known as a color palette. Because of the limitations on the number of bits per picture element (pixel), however, these adapters can only display a subset of the colors in the palette. The color table acts as a map between the pixel value and the color that is displayed. The multi-bit pixel value is used as an index into the color table to generate the actual color on the display. The Set Color Table function allows the virtual terminal to select the set of colors that can be displayed simultaneously when the terminal is active.

This function is valid only when the DSM is in APA or KSR mode. A separate virtual color table is maintained for each mode and for each virtual terminal. The module's mode determines whether this function modifies the APA color table or the KSR color table.

Calling Sequence:

```
cvttstct (color_table, ct_size)
   struct vttcolt *color_table;
   int ct_size;
```

The parameters to Set Color Table are defined as follows:

- `color_table`

  This parameter indicates, in a device-dependent manner, the values used to fill the color table.
- `ct_size`

  This parameter indicates the size in bytes of the color table structure.

Programming Notes:

The size of a color table varies from display to display. Either a default color table or an application-supplied table can be used by this function.

IBM recommends the following approach to coding this entry point:

1. Verify that the number of color entries is exactly correct for your device, otherwise return.
2. Verify that the number of bytes in `ct_size` is exactly correct for your device, otherwise return.
3. If the DSM is active when this entry point is called, copy the supplied color table values to the adapter's color table.

Set Mode

Description: This entry point sets the operation mode of the DSM. Valid modes are:

- Monitored mode (MOM)
- Keyboard send/receive (KSR)
- All-points-addressable (APA)

The mode can be changed at any time, regardless of the state (active or inactive) of the virtual terminal.

The KSR presentation space is saved by the DSM whenever it switches out of KSR mode and is restored when it switches back to KSR mode. If a mode switch occurs when the DSM is in APA mode, the presentation space is not saved by the DSM because the application manages the presentation space in APA mode.

The current mode for a physical display is saved globally and is available to to all instances of the DSM. This value is known as the stored device state and is used by the Activate function to minimize the frequency of device initializations. The current mode for the DSM is also kept for each instance of the module and is known as the stored driver state. The stored driver state value is used by the Set Mode and Activate commands.

To enter monitored mode, the stored driver state of the DSM is set to **not** KSR mode and **not** APA mode. If the module is deactivated, the Set Mode function does not cause the physical device to be relinquished. The Activate function does this (see "Activate" on page 3-143).

To enter KSR mode, the stored driver state of the DSM is set to KSR mode and **not** APA mode. Setting this mode also depends on whether the module is active or inactive. When the module is inactive and the physical display is deactivated, the Set mode function has no effect. When the module is active, the physical display is initialized to KSR mode and the presentation space buffer is drawn to the hardware refresh buffer. In addition, the DSM's KSR mode color table is loaded into the display adapter's real color table.

To enter APA mode, the stored driver state of the DSM is set to APA mode and **not** KSR mode. Setting this mode also depends on whether the module is active or inactive. When the module is inactive and the physical display is deactivated, the Set mode function has no effect. When the module is active, the physical display is initialized to APA mode. In addition, the module's APA mode color table is loaded into the display adapter's real color table. To complete the mode switch to APA mode when the DSM is activated, an Update External Raster Array function must be issued. See "Update External Raster Array" on page 3-173.

Calling Sequence:

```
cvttsetm (adapter_mode)
   int adapter_mode;
```

The parameter to Set Mode is defined as follows:

`adapter_mode`

0 = initialize the adapter in monitored mode.
1 = initialize the adapter in KSR mode.
2 = initialize the adapter in APA mode.

All mode values greater than 2 are reserved.

Programming Notes:

The Set Mode command can be issued for subsequent use by the DSM. For example, you can set the mode to APA mode and deactivate the module. The next time the module is activated, the mode is set to APA.

The default state of the DSM is KSR mode. The Initialize function must be used to set up the presentation space size.

This function can be called from any mode. If you try to set the adapter into an unsupported mode (such as APA mode with a KSR-only display) an error is logged, the function returns, and the adapter state remains unchanged.

IBM recommends the following approach to coding this entry point:

1. If this instance of the virtual terminal is inactive, check the validity of the requested mode. MOM, KSR, and APA are valid modes. Record the mode in the instance-unique static data area.

2. If the virtual terminal instance is active in MOM mode, set the `rscr_mode` field in the real screen table to zero (for MOM mode). Also mark your instance-unique data to MOM mode. If a process is attached to the device driver (`rscr_path[rscr_id-1]` field of the real screen table not equal to zero), detach the process and set the `rscr_path[rscr_id-1]` field to zero.

3. If the virtual terminal instance is active in KSR mode, set the `rscr_mode` field in the real screen table to one (for KSR mode). Also mark your instance-unique data to KSR mode. If a process is not attached to the device driver, attach a process to it. Attach your own process if you are using interrupts, or attach the `vtrm_rc` process if you are not using interrupts. The `vtrm_rc` process ID can be found in the shared resource structure. Set `rscr_path[rscr_id-1]` to the path ID returned from the attach. Next, copy the RAM-resident presentation space to the physical refresh buffer, doing any device-specific raster conversion required. Define or draw the cursor, restoring it to the correct shape and position. Then clear any unused portion of the right side or bottom of the presentation space. Finally, set the physical device's color table to the virtual color table values for the current mode.

4. If the virtual terminal instance is active in APA mode, set the `rscr_mode` field in the real screen table to two (for APA mode). Also mark your instance-unique data

to APA mode. If a process is not attached to the device driver, attach a process to it. Attach your own process if you are using interrupts, or attach the `vtrm_rc` process if you are not using interrupts. The `vtrm_rc` process ID can be found in the shared resource structure. Set `rscr_path[rscr_id-1]` to the path ID returned from the attach.

Update External Raster Array

Description: The external raster array (ERA) is a rectangular array of picture elements in memory that represent the physical screen image of a display. This function displays the ERA on the screen if the virtual terminal is active. If the virtual terminal is inactive, the physical display screen is unaffected by this function.

The width of the ERA always equals the number of picture elements (pixels) that can be shown horizontally on the display. The height of the ERA always equals the number of pixels that can be shown vertically on the display.

The pixel in the upper-left corner of the ERA is always displayed in the upper-left corner of the display screen, and the pixel in the lower-right corner of the ERA is always displayed in the lower-right corner of the display screen.

If the virtual terminal is in KSR mode when this function is called, or if an invalid rectangular area is specified, the results are unpredictable. If the virtual terminal is in APA or monitored mode, the DSM provides no support for cursor.

Calling Sequence:

```
cvttera (screen_pos, era)
   struct vtt_era_rc_parms *screen_pos;
   struct era_data_struc *era;
```

The parameters to Update ERA are defined as follows:

- `screen_pos`

  This parameter specifies the coordinates of the lower-left and upper-right corners of the upright rectangular area of the ERA to display. Note that the actual size of the rectangular area that can be updated by this function is device dependent. Some devices may not support partial byte, halfword, or word updates.

- era

  This parameter represents the ERA structure. The ERA structure is defined as follows:

```
struct era_data_struct
{
  int era_type;
#define VTTPLANE 1
    int rsvd1;
    int era_size;
    short era_lines;
    short era_pels_per_line;
    short era_era_bits_per_pel;
    short rsvd2;
#ifndef ERA_SIZE
    unsigned char era_pixel_values [1];
#else
    unsigned char era_pixel_values [ERA_SIZE];
#endif
} ;
```

  The fields in the preceding structure are defined as follows:

  - era_type

    This field specifies the format of the ERA.

  - VTTPLANE

    This value indicates that the ERA is formatted as a set of one or more planes with one bit of each pixel stored in each of the planes. For example, if the ERA has two bits per pixel, it has two planes.

  - era_size

    This field indicates the number of bytes in the ERA minus one.

  - era_lines

    This value indicates the height of the ERA.

  - era_pels_per_line

    This value indicates the width of the ERA in pels per line.

- `era_era_bits_per_pel`

  This value indicates the number of bits per pixel.

- `era_pixel_values [1]`

  This field indicates the ERA pixel values. See Programming Notes for a definition of ERA pixel values.

- `era_pixel_values [ERA_SIZE]`

  This field indicates the ERA pixel values. See Programming Notes for a definition of ERA pixel values.

Programming Notes:

An ERA's pixel value is stored as a three-dimensional array of 32-bit words. This array has the following structure:

```
pixel_values (bits_per_pixel, vertical_rows, horizontal_columns)
```

The width of the ERA is always a multiple of 32.

This function is valid only in APA mode.

Shared Resource Structure

The shared resource structure contains values that indicate the current virtual terminal configuration. The structure has information of a general nature (resource controller and screen manager ID values, dump table information, path data, and so on) as well as terminal-specific data. The shared resource structure contains information necessary for the resource controller, screen manager, common device utilities and device-specific modules to work together correctly. The structure is made available to the proper components with **_bind** calls issued by the resource controller.

The modules that comprise the shared resource structure are defined on the following pages.

Common Area

This module contains information of a general nature that is required by all components.

```
struct common_area
{
   unsigned int   rs_lock;
   unsigned int   rs_rc_process;
   unsigned int   rs_vtrm_mid;
   unsigned int   rs_sm_process;
   unsigned int   rs_sm_to_kdd_path;
   unsigned int   rs_sm_to_ldd_path;
   unsigned int   rs_kbd_did;
   unsigned int   rs_loc_did;
   short          rs_loc_resolution;
   unsigned char  rs_loc_type;
   unsigned char  rsvd;
   unsigned int   rsvd;
   unsigned int   rs__snd_did;
   short int      rs_active_vt;
   short          rs_connected_vms;
   unsigned int   rs_rsvd;
   unsigned       rs_resv2[8];
};
```

The fields in the common area are defined as follows:

- `rs_lock` is the semaphore that locks the resource structure for a user accessing the various tables.
- `rs_rc_process` is the process ID of the resource controller.

- rs_vtrm_mid is the module ID of the resource controller.
- rs_sm_process is the process ID of the screen manager.
- rs_sm_to_kdd_path is the ID of the path from the screen manager to the keyboard device driver.
- rs_sm_to_ldd_path is the ID of the path from the screen manager to the locator device driver.
- rs_kbd_did is the device ID of the keyboard device driver.
- rs_loc_did is the device ID of the locator device driver.
- rs_loc_resolution is the resolution of the locator device.
- rs_loc_type is the report mode of the locator. A zero in this field indicates relative mode; a one indicates absolute mode.
- rs_snd_did is the device ID of the speaker.
- rs_active_vt is the index into the virtual terminal table (see "Virtual Terminal Table" on page 3-178) that indicates the currently active terminal. If no terminals are open, this field will be zero.
- rs_connected_vms is the number of virtual machines connected to the screen manager.

Resource Controller Dump Table

This module tells the VRM dump function whether to dump the shared resource structure or the resource controller environment when the dump table fills.

```
int         rs_rc_dump_table_length;
struct      dump_table_entry
{
   char             struct_name[8];
   int              struct_length;
   char             *struc_pointer;
   unsigned         struc_resv;
}rs_rc_dump_table_entry[2];
```

The fields in the resource controller dump table are defined as follows:

- rs_rc_dump_table_length is the length of the resource controller dump table.
- struct_name is the name of the structure to dump. When this field has a value of RESOURCE, the shared resource structure is dumped. When this field has a value of ENVRNMNT, the resource controller environment is dumped.
- struct_length is the length of the structure to dump.

- struct_pointer is the address of the structure.

Screen Manager Dump Table

This module tells the VRM dump function to dump the screen manager environment when the table is filled.

```
int       rs_sm_dump_table_length;
struct    dump_table_entry  rs_sm_dump_table_entry;
```

The fields in the screen manager dump table module are defined as follows:

- rs_sm_dump_table_length is the length of the screen manager dump table.
- struc_name is the name of the structure to dump. The screen manager environment is dumped when this field equals ENVRNMNT.
- struc_length is the length of the structure.
- struc_pointer is the address of the structure.

Virtual Terminal Table

This module contains information on each virtual terminal, such as the form of its virtual input devices and the resources used by the terminal. The structure for the virtual terminal table is shown on the following page.

```
struct
{
   unsigned                vt_iodn  : 16;
   unsigned                vt_route_kbd : 1;
   unsigned                vt_route_loc : 1;
   unsigned                vt_kbd_mode : 1;
   unsigned                vt_gate_deactivate : 1;
   unsigned                vt_not_copied : 1;
   unsigned                vt_resrvd       : 3;
   unsigned                vt_led_states : 8;
   int                     vt_rscr_index;
   short                   vt_loc_threshold[2];
   unsigned int            vt_queue_id;
   unsigned int            vt_queue_ecb;
   unsigned int            vt_process_id;
   unsigned int            vt_kbd_input_path;
   unsigned int            vt_loc_input_path;
   unsigned int            vt_kbd_config_path;
   unsigned int            vt_loc_config_path;
   unsigned int            vt_vtmp_module;
   unsigned int            vt_vdd_module;
   unsigned int            vt_rsvd;
   int                     vt_dump_table_length;
#ifndef                 NUM_VTS
#define                 NUM_VTS 32
#endif
   struct   dump_table_entry   vt_dump_table_entry[5];
   unsigned int vt_resrvd2;
} vt_table_entry[NUM_VTS];
```

The fields in the virtual terminal table are defined as follows:

- vt_iodn is the virtual terminal IODN. A queue ID of zero indicates that this table entry is not currently assigned.

The next five fields are flags that indicate the input devices requested by the virtual terminal, the form of keyboard input required, and the gating for the deactivate request.

- vt_route_kbd indicates whether the virtual terminal receives keyboard input. If this flag is set (equals one), the virtual terminal receives keyboard input.

- vt_route_loc indicates whether the virtual terminal receives locator input. If this flag is set (equals one), the virtual terminal receives locator input.
- vt_kbd_mode indicates the reporting mode of the virtual keyboard for an inactive terminal. The keyboard device driver maintains the current reporting mode of the active virtual terminal.
- vt_gate_deactivate indicates whether the virtual terminal can accept the Deactivate command from the screen manager. If this flag is set (equals one), the screen manager will send the Deactivate request to the virtual terminal at the proper time.
- vt_not_copied indicates whether the VTMP module was copied at open time. If this flag is set (equals 1), the VTMP was copied and must be deleted when the virtual terminal is closed.
- vt_led_states indicates the state of the LEDs on the virtual keyboard for an inactive terminal. When this field equals one, an LED is on. The keyboard device driver maintains the current LED states for the active virtual terminal.
- vt_rscr_index is the index into the real screen table (see "Real Screen Table" on page 3-181) for the real display currently used by the virtual terminal.
- vt_loc_threshold contains the horizontal (1) and vertical (2) thresholds for reporting locator input.

The next two fields contain information about the queue that is serviced by the virtual terminal's VTMP.

- vt_queue_id is the queue ID.
- vt_queue_ecb is the ECB mask that is posted when a queue element is placed on the queue.
- vt_process_id is the process ID of the virtual terminal process.
- vt_kbd_input_path is the ID of the path from the keyboard device driver to the virtual terminal.
- vt_loc_input_path is the ID of the path from the locator device driver to the virtual terminal.
- vt_kbd_config_path is the ID of the path from the virtual terminal to the keyboard device driver.
- vt_loc_config_path is the ID of the path from the virtual terminal to the locator device driver.
- vt_vtmp_module is the module ID of the virtual terminal's VTMP.
- vt_vdd_module is the module ID of the virtual terminal's device-specific module.
- vt_dump_table_length is the length of the virtual terminal's dump table.
- struc_name is the name of the structure to dump. The following names are valid:

  INITPARM – the initialization structure for the VTMP.

VTMSTRUC – the VTMP environment.
LINE_DAT – the VTMP tab rack.
VTTnENV – the device-specific module environment; the n identifies the adapter/monitor combination.
VTTnPS – the device-specific module's presentation space when not included with the DSM environment dump; the n identifies the adapter/monitor combination.

- struc_length is the length of the structure.
- struc_pointer is the address of the structure.

Real Screen Table

This module contains information about the display device driver's physical screen and display adapter.

```
struct
{
   unsigned int      rscr_ident;
   int               rscr_memory;
   unsigned int      rscr_device_id;
   unsigned short    rscr_iodn;
   unsigned char     rscr_mode;
   unsigned char     rscr_usage : 8;
   unsigned          rscr_last_active_vt : 16;
   unsigned          rscr_default : 1;
   unsigned          rscr_resrvd : 7;
   unsigned          rscr_vdd_type : 8;
   unsigned int      rscr_module;
   unsigned int      rscr_path;
   unsigned int      rscr_free_area[16];
   int               rscr_dump_table_length;
   struct  dump_table_entry      rscr_dump_table_entry;
#ifndef                NUM_RSCR
#define                NUM_RSCR 4
#endif
}   rscr_table_entry  [NUM_RSCR];
```

The fields in the Real Screen Table module are defined as follows:

- rscr_ident is a real screen's external identifier, by which application programs refer to the real screen. A zero screen ID indicates that the real screen is currently unassigned.

- rscr_memory is the size of the memory allocated for an adapter's frame buffer. This field, which is set by the common device utilities module, is valid only for adapters dependent on this data.
- rscr_device_id is the device ID of the display device driver's real screen.
- rscr_iodn is the IODN of the display device driver's real screen.
- rscr_mode indicates the current mode of the adapter for the specified real screen.
- rscr_usage is a value indicating the number of virtual terminals using the specified real screen. The real screen is not removed from the configuration if any virtual terminal is using it.
- rscr_last_active_vt is the index for the last virtual terminal that was using this screen.
- rscr_default indicates when set (equals one) that this display is the default display for virtual terminals when opened.
- rscr_vdd_type indicates the implementation language of the device-specific module. For the C language, this value is one.
- rscr_module is the module ID of the device-specific module for the specified real screen.
- rscr_path is the path ID from the the display device driver to either a virtual terminal or the resource controller (for when a virtual terminal is being closed).
- rscr_free_area is an area allocated for free device-dependent usage by instances of DSMs for this real screen.
- rscr_dump_table_length is the length of dump table for the real screen.
- struc_name is the name of the structure to dump. For character-oriented adapters, this name is VTTnDISP, where n identifies the adapter/monitor combination.
- struc_length is the length of the dumped structure.
- struc_pointer is the address of the dumped structure.

Real Font Table

This module contains descriptive and addressability information about each available font.

```
struct
{
   short            rfnt_ident;
   short            rfnt_class;
   int              rfnt_font_style;
   int              rfnt_font_attr;
   short            rfnt_size [2];
   short            rfnt_capline;
   short            rfnt_baseline;
   short            rfnt_ul_top.
   short            rfnt_ul_bot.
   unsigned short   rfnt_resrvd;
   unsigned short   rfnt_fonttab_segid;
   unsigned int     rfnt_fonttab_baseaddr;
   int              rfnt_fonttab_size;
#ifndef          NUM_RFNT
#define          NUM_RFNT  32
#endif
}rfnt_table_entry[NUM_RFNT];
}extern struct common_area vttrs;
```

The fields in the Real Font Table are defined as follows:

- `rfnt_ident` is the external identifier of the real font. A zero ID indicates that this real font entry is unassigned.
- `rfnt_class` identifies the class of displays that can display this font type.
- `rfnt_font_style` is the style of the font.
- `rfnt_font_attr` is the attribute of the font.
- `rfnt_size` is the horizontal (0) and vertical (1) size of the font in device coordinates (for monospace font) or the horizontal and vertical size of the largest character in the font (variable space font).
- `rfnt_capline` is the row number of the font's capline.
- `rfnt_baseline` is the row number of the font's baseline.
- `rfnt_ul_top` is the row number of the top scan line in the underscore.

- `rfnt_ul_bot` is the row number of the bottom scan line in the underscore.

The following fields contain addressing information for device-specific modules to access the font.

- `rfnt_fonttab_segid` is the segment ID for the area that contains the font.
- `rfnt_fonttab_baseaddr` is the base address for the area that contains the font.
- `rfnt_fonttab_size` is the size of the font.

Define Color Table

Description: This is a service routine referenced by the Initialize and Finis entry points that allows the DSM to define the two default color tables to the VTMP. These color tables can be modified with the Set Color Table function (see "Set Color Table" on page 3-169).

Note that this is **not** an entry point to a device-specific module.

The DSM must define two default color tables, one for character (KSR) displays and one for APA displays, so the VTMP can use the Set Color Table function to return the terminal to default state. Define Color Table is called by the Initialize function with address and length values for the color tables. The Finis function calls Define Color Table with address and length values of zero to release the tables before the device-specific module is terminated.

Calling Sequence:

```
cvttdct (color_table_k, ct_size_k, color_table_a, ct_size_a)
   struct vttcolt *color_table_k;
   int ct_size_k;
   struct vttcolt *color_table_a;
   int ct_size_a;
```

The parameters to Define Color Table are defined as follows:

- `color_table_k` and `color_table_a` specify the values used to fill the color table for the KSR and APA defaults, respectively. The structures pointed to by this parameter must be static, as no copies are made. These structures have the following format:

  ```
  struct   vttcolt
  {
     int ct_num;
  #ifndef COLORTAB
     unsigned int ct_entry[1];
  #else
     unsigned int ct_entry_COLORTAB;
  #endif
  } ;
  ```

  The fields in the preceding structure are defined as follows:

  - `ct_num` is the number of color table entries for a device. This is a fixed, device-dependent value.

- `ct_entry` is an entry in the color table array. The number of entries in an array varies between devices, but is fixed for a particular device.
- `ct_entry_COLORTAB` is an entry in the color table array. The number of entries in an array varies between devices, but is fixed for a particular device.

- `ct_size_k` and `ct_size_a` specify the size in bytes of the KSR and APA color table structures, respectively.

Programming Notes:

The size of the color table varies from device to device.

Default Color Table Structures

Two default color tables, one for KSR and one for APA mode, are provided for all supported display types. These tables allow you to establish an ANSI 3.64-compatible color definition for KSR mode or a PC-compatible color definition for APA and coprocessor modes. The size and organization of color tables is strictly device-dependent. The tables are initialized with values that establish the following colors for the indicated display.

Display Type	Color Table Index	Hex Value	KSR Mode	APA Mode
PC Mono A/N	1	0	Black	Black
	2	2	Green	Green
APA-8	1	0	Black	Black
	2	7	White	White
EGA/PC Mono	1	0	Black	Black
	2	2	Green	Green
EGA/Enhanced Color Display	1	00	Black	Black
	2	04	Red	Blue
	3	02	Green	Green
	4	3E	Yellow	Cyan
	5	01	Blue	Red
	6	05	Magenta	Magenta
	7	03	Cyan	Brown
	8	07	White	White
	9	38	Gray	Gray

Figure 3-38 (Part 1 of 3). Default Color Tables

Display Type	Color Table Index	Hex Value	KSR Mode	APA Mode
	10	3C	Light red	Light blue
	11	3A	Light green	Light green
	12	06	Brown	Light cyan
	13	39	Light blue	Light red
	14	3D	Light magenta	Light magenta
	15	3B	Light cyan	Yellow
	16	3F	High intensity white	High intensity white
APA-8 Color	1	00	Black	Black
	2	02	Red	Blue
	3	08	Green	Green
	4	0F	Yellow	Cyan
	5	20	Blue	Red
	6	22	Magenta	Magenta
	7	28	Cyan	Brown
	8	2A	White	White
	9	15	Gray	Gray
	10	17	Light red	Light blue
	11	1D	Light green	Light green
	12	05	Brown	Light cyan
	13	35	Light blue	Light red
	14	37	Light magenta	Light magenta
	15	3D	Light cyan	Yellow
	16	3F	High intensity white	High intensity white
PC Color Display	1	00	Black	Black
	2	01	Blue	Blue
	3	02	Green	Green
	4	03	Cyan	Cyan
	5	04	Red	Red
	6	05	Magenta	Magenta
	7	06	Brown	Brown
	8	07	White	White
	9	08	Gray	Gray
	10	09	Light blue	Light blue

Figure 3-38 (Part 2 of 3). Default Color Tables

Display Type	Color Table Index	Hex Value	KSR Mode	APA Mode
	11	0A	Light green	Light green
	12	0B	Light cyan	Light cyan
	13	0C	Light red	Light red
	14	0D	Light magenta	Light magenta
	15	0E	Yellow	Yellow
	16	0F	High intensity white	High intensity white
APA-16	1	0	Black	Black
	2	2	White	White

Figure 3-38 (Part 3 of 3). Default Color Tables

When the DSM is in KSR mode, the first eight colors in the preceding tables can be used as either foreground or background colors; the last eight colors can only be used as foreground colors.

When the DSM is in APA mode, the color corresponding to a pixel value can be calculated by adding one to the pixel value; the result is the index into the preceding tables.

The default color table for the 5081 Megapel Display is shown in the following table. Note that APA mode is not supported for this display. The colors are formed by combinations of red, green, and blue, indicated in the table as R, G, and B.

Display Type	Color Table Index	Color Value R G B	KSR Mode
5081	1	000000	Black
	2	F00000	Red
	3	00F000	Green
	4	F0F000	Yellow
	5	0000F0	Blue
	6	B000B0	Magenta
	7	00F0F0	Cyan
	8	D0D0D0	White
	9	404040	Gray
	10	F04040	Light red

Figure 3-39 (Part 1 of 2). 5081 Display Default Color Table

Display Type	Color Table Index	Color Value R G B	KSR Mode
	11	70F000	Light green
	12	C08020	Brown
	13	0070F0	Light blue
	14	E030E0	Light magenta
	15	70F0D0	Light cyan
	16	F0F0F0	High intensity white

Figure 3-39 (Part 2 of 2). 5081 Display Default Color Table

Display Device Driver Interface

Although display devices have more layers of software between the operating system and the hardware as compared to other devices (see Figure 3-36 on page 3-140), the actual hardware device drivers are similar for all devices. Of the typical device driver entry points described in "Common Routine Interface" on page 2-14, the display device driver must define the following:

Entry Point	Op Code	Parameters
Define device	0x3C	Define device structure
Initialize device	0x3D	Device ID
Interrupt handler	0x28	Bus interrupt level
I/O initiate	0x1E	Queue element
Terminate device	0x3E	Device ID

Figure 3-40. Display Device Driver Entry Points

Define Device Entry Point

This function disables the adapter's interrupts, accepts a DDS from the VRM, copies the DDS to a static data area, and returns the address of the area to the VRM.

This function is called in the following manner:

```
rc = main(0x3C, def_dev_struct, sizeof_def_dev_struct)
```

The parameters to this entry point are defined as follows:

- `0x3C` is the define device op code.
- `def_dev_struct` is a pointer to the DDS.
- `sizeof_def_dev_struct` is the length of the DDS.

The DDS used by a display device device driver has the same three parts (header, hardware characteristics section, and device characteristics section) as a DDS for any other device. Note that display device drivers do not have error log sections. The header has the fields shown in Figure 2-2 on page 2-5.

Header fields defined by IBM for supported displays include:

- Display IODNs:
  PC Monochrome A/N - 0x100
  APA-8 - 0x101
  EGA/PC Monochrome - 0x102
  EGA/Enhanced Color - 0x103
  APA-16 - 0x104
  APA-8 Color - 0x105
  5081 - 0x107

- IOCN of hardware display device driver:
  PC Monochrome A/N - 0x201
  APA-8 - 0x202
  EGA/PC Monochrome - 0x214
  EGA/Enhanced Color - 0x214
  APA-16 - 0x215
  APA-8 Color - 0x216
  5081 - 0x218

The format of the hardware characteristics fields are shown in Figure 2-3 on page 2-6. Note that the APA-16 and 5081 display adapters are the only display adapters that support DMA. As such, the high-order bit of the DMA Type field is set to 1. In addition, the alternate DMA bit is set to define these adapters as alternate DMA devices. The DMA channel number field is set to 7 for the APA-16 and 5081 adapters.

The device characteristics section is defined on the following page.

```
struct vtt_dds_dev_char
{
   int              dev_length;
   unsigned int     dev_rdev_id;
   unsigned short   dev_vdd_iocn;
   short            dev_iocn_rsvd;
   int              dev_monitor;
   int              dev_adapter_memory;
   short            dev_font_class;
   short            dev_font_rsvd;
   int              dev_font_width;
   int              dev_font_height;
   short            dev_avt_format;
   short            dev_avt_rsvd;
   int              dev_bits_per_pel;
   unsigned         dev_monitor_type    :1;
   unsigned         dev_vlt_modify      :1;
   unsigned         dev_adapter_status  :1;
   unsigned                                 :29;
   int              dev_num_mode_struc;
   struct
   {
      unsigned         dev_apa       :1;
      unsigned         dev_blink     :1;
      unsigned                       :30;
      int              dev_width_pel;
      int              dev_height_pel;
      int              dev_width_mm;
      int              dev_height_mm;
      unsigned         dev_color_ref;
#ifndef   DEVNMMOD
#define   DEVNMMOD1
#endif
   }  dev_mode  [DEVNMMOD];
};
```

The device characteristics fields of a display device driver are defined as follows:

- dev_length is the length of the device characteristics section.

- dev_rdev_id is the ID of the adapter/monitor combination. The most significant byte of this field always equals 0x04, and the ID is in the next byte. The least significant halfword always equals 0x00. Values < 128 are reserved. Values defined by IBM include:

  0x04010000 – PC Monochrome A/N
  0x04020000 – APA-8
  0x04030000 – EGA/PC Monochrome
  0x04040000 – EGA/Enhanced Color
  0x04050000 – APA-16
  0x04060000 – APA-8 Color
  0x04080000 – 5081 display
  0x04070000 – Sample PC Color

  This ID accompanies the sample display device driver that can be found in the AIX directory **/usr/lib/samples**.

- dev_vdd_iocn contains the IOCN of the device-specific module that runs with the adapter/monitor combination. This value must match the IOCN value established for the module with the **Define Code SVC**. Values defined by IBM include:

  0x00A1 – PC Monochrome A/N
  0x00A2 – APA-8
  0x00A3 – EGA/PC Monochrome
  0x00A4 – EGA/Enhanced Color
  0x00A5 – APA-16
  0x00A6 – APA-8 Color
  0x00A8 – 5081 display
  0x00AF – Sample PC Color.

- dev_monitor is the device ID of the display monitor. IDs defined by third parties (not IBM) should be >256. Values defined by IBM include:

  0x0000 – PC Monochrome A/N
  0x0001 – APA-8
  0x0000 – EGA/PC Monochrome
  0x0002 – EGA/Enhanced Color

  Note that the lposts determine which monitor is attached to the EGA adapter by reading the on-card switches.
  0x0004 – APA-16
  0x0003 – APA-8 Color
  0x0005 – Sample PC Color
  0x0006 – 5081 display.

- dev_adapter_memory indicates the size in bytes of the adapter's refresh buffer. Values defined by IBM include:

  PC Monochrome A/N - 4K bytes
  EGA (both) - 64K bytes, expandable to 128K or 256K with Graphics Memory Expansion Adapter

APA-8 - 64K bytes
APA-8 Color - 256K bytes
APA-16 - 128K bytes
5081 - 256K bytes.

- dev_font_class indicates the class of font that this device can use. Three classes are defined, APA display fonts (set to one), KSR display fonts (set to zero), and 5081 display fonts (set to two). Font values defined by third parties (not IBM) should be > 256.
- dev_font_width is a value used only by KSR displays that specifies the width of the character box in pixels. Values defined for IBM adapters are defined as follows:

  PC Monochrome A/N – 9 pixels
  EGA/Color – 8 pixels
  EGA/PC Monochrome – 9 pixels
  PC Color sample – 8 pixels
  5081 display – 8 pixels
  APA adapters set this field to zero.

- dev_font_height is a value used only by KSR displays that specifies the height of the character box in pixels. The height values defined by IBM for the PC Monochrome and EGA are 14 pixels. The PC Color sample and the 5081 display have a height value of 8 pixels. APA adapters set this field equal to zero.
- dev_avt_format indicates, for APA adapters only, the format of the APA data buffer. Two formats are supported. When this value is set to one, a raster plane data format is specified. This means that the raster array must be formatted as a set of one or more bit planes with one bit of each pixel value stored in each of the bit planes. The number of bit planes equals dev_bits_per_pel. When this value is set to zero, the APA terminal mode is unsupported. Only the APA-8, APA-16, and APA-8 Color adapters support APA mode.
- dev_bits_per_pel indicates the total number of bits in each pixel value. This value is always set to one for adapters that run in KSR mode. For the APA adapters, the APA-8 and APA-16 adapters have one bit per pixel, and the APA-8 Color adapter has four bits per pixel. The 5081 display has eight bits per pixel.
- dev_monitor_type indicates whether the monitor supports color. A one indicates a color monitor; a zero indicates a monochrome monitor.
- dev_vlt_modify indicates whether an adapter's raster array can be modified. Of the IBM-supported adapters, the APA-8 Color, EGA/Enhanced Color, and 5081 adapters have modifiable raster arrays.
- dev_adapter_status is a value filled in by the adapter's loadable POST that indicates if the adapter is working properly. If this value equals one, the adapter is working properly; a zero indicates the adapter is not working properly.
- dev_num_mode_struc specifies the number of modes supported by the adapter. If the adapter supports only APA or KSR mode, this value equals one. It is set to two for adapters that can

support both modes (such as the EGA/PC Monochrome and EGA/Enhanced Color adapters). If the adapter supports two modes, the first element in the structure describes the KSR characteristics of the adapter and the second element describes the APA characteristics.

- dev_apa is set to one if the mode structure describes an APA mode and to zero if the structure describes a KSR mode.
- dev_blink is set to one if the adapter supports blinking characters. The KSR mode adapters support blinking and the APA adapters do not support blinking.
- dev_width_pel indicates the width of the displayable presentation surface in pixels. All IBM-supplied adapters are defined as 720 pixels wide except the EGA/Enhanced Color display, which is set at 640 pixels, and the APA-16 and 5081 adapters, which are set at 1024 pixels.
- dev_height_pel indicates the height of the displayable presentation surface in pixels. The KSR-mode adapters are defined as 350 pixels high. The APA-8 adapters are defined as 512 pixels, and the APA-16 adapter is 768 pixels. The 5081 display adapter is defined as 1024 pixels.
- dev_width_mm indicates the width of the displayable presentation surface in millimeters. Values defined by IBM include:

  PC Monochrome A/N – 204 mm.
  EGA/PC Monochrome – 204 mm.
  EGA/Enhanced Color – 240 mm.
  APA-8 – 210 mm.
  APA-8 Color – 240 mm.
  APA-16 – 267 mm.
  5081 – 284 mm.

- dev_height_mm indicates the height of the displayable presentation surface in millimeters. Values defined by IBM include:

  PC Monochrome A/N – 135 mm.
  EGA/PC Monochrome – 135 mm.
  EGA/Enhanced Color – 160 mm.
  APA-8 – 149 mm.
  APA-8 Color – 171 mm.
  APA-16 – 203 mm.
  5081 – 284 mm.

- dev_color_ref contains the offset in bytes from the beginning of vttdev_char_struct to the following structure:

```
struct       dev_color_struct
{
   int        dev_color_total;
   int        dev_color_active;
   int        dev_color_fg;
   int        dev_color_bg;
   #ifndef    DEVNMCOL;
   #define    DEVNMCOL 1;
   #endif;
   int        dev_color_value  [DEVNMCOL];
}
```

The fields in the preceding structure are defined as follows:

- dev_color_total contains the total number of real colors that can be displayed (not necessarily simultaneously). The PC Monochrome A/N, EGA/PC Monochrome, APA-16 and APA-8 can display two colors; the APA-8 Color and EGA/Enhanced Color can display 64 colors. The 5081 display can display 4,096 colors.
- dev_color_active indicates the number of colors that can be displayed simultaneously. The PC Monochrome A/N, EGA/PC Monochrome, APA-16 and APA-8 can display two colors simultaneously; the APA-8 Color and EGA/Enhanced Color can display 16 colors simultaneously. The 5081 display can display 256 colors simultaneously.
- dev_color_fg indicates the number of foreground colors that can be displayed simultaneously. The PC Monochrome A/N, EGA/PC Monochrome, APA-16 and APA-8 can display two foreground colors simultaneously; the APA-8 Color and EGA/Enhanced Color can display 16 foreground colors simultaneously. The 5081 display can display 256 colors simultaneously.
- dev_color_bg indicates the number of background colors that can be displayed simultaneously. The PC Monochrome A/N, EGA/PC Monochrome, APA-16 and APA-8 can display two background colors simultaneously; the APA-8 Color and EGA/Enhanced Color can display 16 background colors simultaneously. The 5081 display can display 256 colors simultaneously.
- dev_color is an array that specifies the colors that can be displayed simultaneously for a given device. For example, the APA-8 Color adapter can display 16 colors simultaneously, so the number of elements in its array is 16.
- dev_color_val is a hexadecimal value from 0 to dev_color_total - 1 that indicates the color to display. The color that corresponds to a particular dev_color_val is device dependent. A table with the colors and corresponding hex values for the particular adapters is shown in Figure 3-38 on page 3-186.

Initialize Device Entry Point

This function activates the display adapter. The adapter's control registers are initialized with information from the DDS, the frame buffer is cleared, and the device driver's queue ID is saved in a static data area so any queue elements sent to the driver can be dequeued. For the EGA adapter, the RAM character generator is also loaded.

This function is called in the following manner:

```
rc = main(0x3D, init_parms_struct, sizeof_init_parms_struct)
```

The parameters to this entry point are defined as follows:

- 0x3D is the initialize device op code.
- init_parms_struct is a pointer to the structure shown on the following page.

```
struct
{
   unsigned int device_id;
   unsigned short iodn;
   short rsvd;
   unsigned int queue_id;
   unsigned int module_id;
   unsigned int slih_id;
} init_parms;
```

- `sizeof_init_parms_struct` is the length of the structure.

Interrupt Handler Entry Point

This function processes interrupts generated by the display adapter. Since display adapters should never cause an interrupt under normal circumstances, this entry point is called by a FLIH when the display adapter shares a hardware bus interrupt level with another device.

When called, this entry point does the following:

- Determines if its display adapter caused the interrupt.
- If the adapter did cause the interrupt, the interrupt handler disables the interrupt and returns a zero to the FLIH, indicating successful processing of the interrupt.
- If the adapter did not cause the interrupt, the interrupt handler returns a two to the FLIH, informing the FLIH to continue to search other devices on the same bus interrupt level.

This function is called in the following manner:

```
rc = main(0x28, int_level, reserved)
```

The parameters to this entry point are defined as follows:

- `0x28` is the interrupt handler op code.
- `int_level` is a pointer to an integer that contains the hardware bus interrupt level.
- The length parameter is not used by this entry point.

Initiate I/O Entry Point

This function removes a queue element from the device driver's queue. When a process detaches from the device driver's queue, the VRM enqueues an element in the driver's queue. This function does nothing more than dequeue that queue element.

This function is called in the following manner:

```
rc = main(0x1E, reserved, reserved)
```

The parameters to this entry point are defined as follows:

- 0x1E is the initiate I/O op code.
- The data parameter is not used by this entry point.
- The length parameter is not used by this entry point.

Terminate Device Entry Point

This function disables the display adapter and its interrupts and is called in the following manner:

```
rc = main(0x3E, device_id, reserved)
```

The parameters to this entry point are defined as follows:

- 0x3E is the terminate device op code.
- device_id is a pointer to the 32-bit device ID that is being disabled.
- The length parameter is not used by this entry point.

Coding Concepts for Adapters that Generate Interrupts

Display adapters that generate interrupts (such as the IBM 5081 Megapel adapter) have special coding requirements in order to utilize the interrupts in the RT PC. These requirements consist of additional code or function in certain entry points of the device-specific module (described in "Display Device Driver Considerations" on page 3-140) and the display device driver (described in "Display Device Driver Interface" on page 3-190).

Device-Specific Module Requirements

Some of the entry points of device-specific modules require additional code or function to utilize adapter-generated interrupts. These entry points are described in the following sections.

Activate

If you are activating a monitored mode virtual terminal, first perform any **_detach** operations based on the `rscr_path` count. Then issue **_attach** calls to attach the virtual terminal's VTMP to the device driver and the device driver to the VTMP. Next, issue an **_enque** with a general purpose queue element to the device driver, specifying the VTMP-to-device driver path returned from the **_attach**.

Deactivate and Finis

If your virtual terminal is in monitored mode and you want to perform the deactivate or finis routines, issue **_detach** calls, specifying the IDs of the paths from the device driver to the VTMP and the VTMP to the device driver.

Set Mode

If you are switching to a monitored mode virtual terminal, first perform any **_detach** operations based on the `rscr_path` count. Then issue **_attach** calls to attach the virtual terminal's VTMP to the device driver and the device driver to the VTMP.

Next, issue an **_enque** with a general purpose queue element to the device driver, specifying the VTMP-to-device driver path returned from the **_attach**.

When switching to a KSR mode virtual terminal, issue **_detach** calls, specifying the IDs of the paths from the device driver to the VTMP and the VTMP to the device driver.

Display Device Driver Requirements

Some of the entry points of display device driver modules require additional code or function to utilize adapter-generated interrupts. These entry points are described in the following sections.

Initialize and Terminate

At both initialize and terminate times, the path ID of the device driver-to-VTMP field should be set to 0. Also, interrupts should be disabled if the display adapter is able to do so.

Interrupt Handler

If the device driver-to-VTMP path ID is not set to zero, **_enque** a general purpose queue element on the device driver-to-VTMP path. The operation options of this queue element must be in the range 0x01 to 0x0B. As many as 20 bytes of data can be sent in a general purpose queue element to an application by way of a monitor mode input ring using these display adapter-generated interrupts.

Initiate I/O

You must modify this entry point to handle the queue elements (generated by the activate and set mode routines of a device-specific module) that inform the device driver of a new device driver-to-VTMP path ID. The initiate I/O routine must update the path ID and enable interrupts if the display adapter is capable of doing so.

Virtual Terminal Resource Manager

The virtual terminal resource manager is a ***virtual device*** manager that controls the display screen and handles configuration and initialization of physical terminal components. A virtual device is typically one or more VRM processes that execute in a more privileged environment than a device driver. For example, a virtual device can page fault, can issue wait operations, and has access to all VRM services.

The command interface and return status information for the VTRM is described in "Virtual Machine Interface to the VTRM" on page 3-6.

The following section describes the fields in the define device structure of the VTRM. This information is useful if you want to change any of the virtual terminal default parameters. Also, if you issue a **Query Device SVC**, you need to know the definitions of the device characteristics fields. Note that the VTRM DDS has no hardware characteristics or error log section.

Figure 3-41 on page 3-203 shows the format of the VTRM DDS header and device characteristics section. The fields are described in the section that follows.

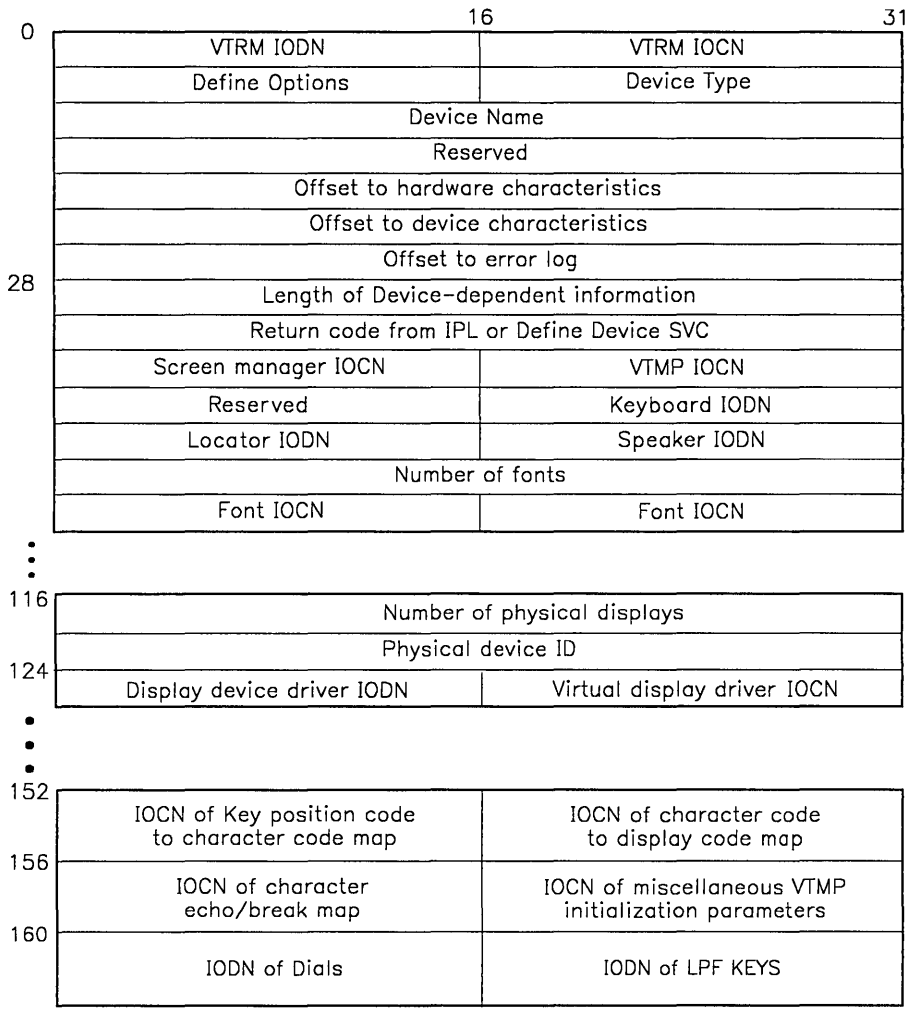


Figure 3-41. VTRM Define Device Structure

The first 28 bytes of the DDS make up the DDS header. Values of particular interest include:

- VTRM IODN = 0x0205
- VTRM IOCN = 0x0082
- Options = not applicable
- Device type = 0x0002 (shared device)
- Device name = 'VTRM'A
- Offsets:

To hardware characteristics = 0
To device characteristics = 28
To error log = 0.

The device characteristics fields are defined as follows:

- Length of device-dependent information = 33 words
- Return code from IPL = 0 for successful IPL, 6499 for unsuccessful IPL
- Screen manager IOCN = 0x0084
- VTMP IOCN = 0x0081
- Keyboard IODN = filled in at IPL time
- Locator IODN = filled in at IPL time

  The locator device is optional. If the locator is not used, this field should be set equal to zero.
- Speaker IODN = filled in at IPL time

  The speaker device is optional. If the speaker is not used, this field should be set equal to zero.
- Number of fonts

  At IPL time, this value is 1. However, you can define up to 31 additional fonts. After IPL time, this field reflects the actual number of configured fonts.
- Font IOCN = 0x0101 for default font, 0x0108 for the 5081 font

  This halfword is repeated 31 times to accomodate the IOCNs of fonts you may define. Initially, these 31 halfwords are set equal to zero.
- Number of physical displays = filled in at IPL time

  The VRM supports as many as four physical displays. The next three fields (physical display identifier, display device driver IODN, virtual display driver IOCN) are repeated four times to accommodate additional displays. Unused fields in this array are set equal to zero.
- Physical display identifier

  This value is a code which identifies the particular display adapter/monitor combination.
- Display device driver IODN

  This value represents the IODN for the driver of the physical display device.
- Virtual display driver IOCN

  This value represents the IOCN for the virtual display driver.
- IOCN of key position code to character code map

  This field represents the IOCN of the code that currently defines how key position codes map to character codes. This value is 0xC0.

- IOCN of character code to display code map

  This field represents the IOCN of the code that currently defines how character codes map to display codes (the current font). This value is 0xC2.

- IOCN of echo/break map

  This field represents the IOCN of the code that currently defines the character echo and break maps. This value is 0xC1.

- IOCN of VTMP initialization parameters.

  This field represents the IOCN of the code that currently defines the VTMP initialization parameters. Examples of these parameters include protocol modes, tab rack, and so on. This value is 0xC3.

- IODN of Dials

  This field represents the IODN for the IBM 5080 Dials Feature. The value is defined as 0x39.

- IODN of LPF Keys

  This field represents the IODN of the IBM 5080 Lighted Program Function Keyboard Feature. The value is defined as 0x3A.

Chapter 4. Block I/O Subsystem

CONTENTS

About This Chapter

This chapter describes the block I/O subsystem of the VRM. The block I/O subsystem includes a device manager, device drivers, and VRM or operating system processes that work together in a variety of networks to transfer blocks of data.

Block I/O Device Driver Considerations

This section describes the interfaces necessary to code a block I/O device driver. A block I/O device driver is a component of the block I/O subsystem. Other subsystem components include a block I/O device manager (described in "Block I/O Device Manager" on page 4-38), logical link control processes, and hardware adapters. The role of the device drivers in this subsystem is to service hardware interrupts and control the communications and local area network adapters. "Baseband Device Driver" on page 4-47 describes a specific driver in this subsystem.

The components of the block I/O subsystem are shown in the following figure.

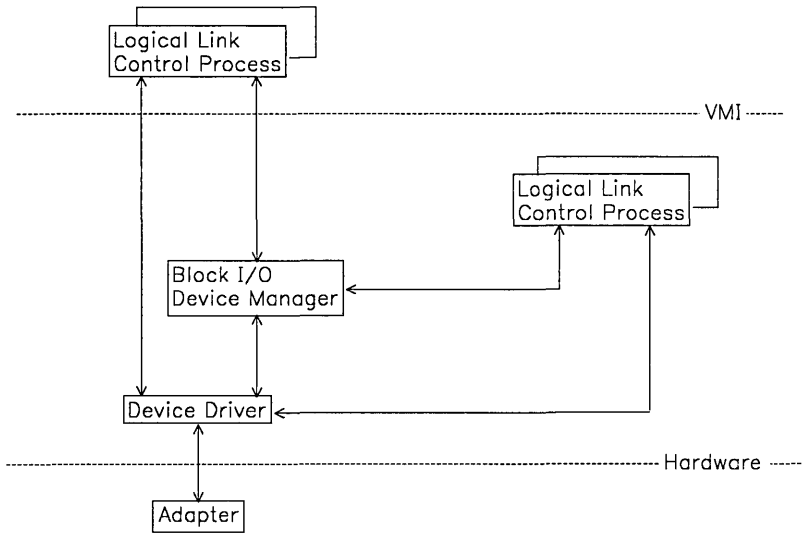


Figure 4-1. Block I/O Subsystem Components

The device driver module consists of the entry points described in "Common Routine Interface" on page 2-14. Specific processing done by some of the entry points of a block I/O device driver includes:

- Define device — this routine is called once for each port or device the device driver controls. Each port or device has a unique define device structure. When the driver is defined, it uses the **_dalct** to obtain an area of memory large enough to contain the DDS, network ID correlation table, and SLIH ring queue.
- Initialize device — in this routine the driver defines its check parameters, I/O initiate, interrupt handler, off-level interrupt handler, and exception handler routines. The driver may also initialize its data areas and adapter in this routine, or in the I/O initiate routine as part of the processing performed when it receives an **open** command from the device manager.

- I/O initiate — a block I/O device driver must support the **open** command from the device manager, as well as the **start device, halt device,** and **write** commands from an LLC process. Device-dependent commands may also be supported.
- Interrupt handlers — the device driver must determine whether its adapter (or which port on an adapter that supports multiple ports) caused the interrupt. The second-level interrupt handler makes this determination and, if its port or adapter caused the interrupt, performs the minimum amount of processing necessary to clear the interrupt. The off-level interrupt handler routine can then be used to perform any additional processing off of the interrupt level. The off-level routines should be set up to handle as much of the processing as possible, leaving the SLIH path as short as possible.

Block I/O Programming Interfaces

The block I/O subsystem has three programming interfaces. They are:

- LLC process to block I/O device manager (described in "Block I/O Device Manager" on page 4-38)
- Block I/O device manager to device driver
- LLC process to device driver.

The device driver interfaces are described in the following sections.

Block I/O Device Manager-to-Device Driver Interface

Because both the device manager and the device drivers are contained within the VRM, only the following runtime routines are required for communication:

- **_attchq**

  When the device manager receives the first **open** command from an LLC for a device driver's IODN, the device manager must first attach itself to the device driver. The device manager uses the **_attchq** routine with the following parameters:
  - From ID — the process ID of the device manager
  - To ID — the device ID of the device driver
  - Acknowledge parameters — indicates a long acknowledge type, the QID of the device manager's receive queue as the Return QID, an enqueue priority of zero, and a maximum interrupt depth of zero.

  Because this is the first attach to the device driver, the driver's Initialize Device routine is called as a result of the attach.

  The returned information from the **_attchq** indicates the ID of the device manager-to-device driver path.

After the device manager attaches to the device driver, the device manager sets up a path between the LLC process and the device driver with another **_attchq**. The parameters to this call are as follows:

- From ID — the process ID of the LLC
- To ID — the device ID of the device driver
- Acknowledge parameters — depends on whether the LLC is in the VRM or operating system space.

  For VRM LLCs, the long acknowledge type, the Return QID, and the LLC's requested maximum interrupt depth are specified. The enqueue priority is set to zero.

  For operating system LLCs, the interrupt type and LLC's requested interrupt level/sublevel are specified. The maximum interrupt depth is set to 15 and the return QID is set to zero.

In order for the device manager to keep track of the total number of IODNs opened (22 is the maximum), as well as to determine when to detach from the various IODNs, the device manager maintains an internal table of IODNs. The table is fixed in size at 22 entries. If there is no unused entry in the table, the device manager returns an error for the **open** request. The structure of this table is shown in Figure 4-2.

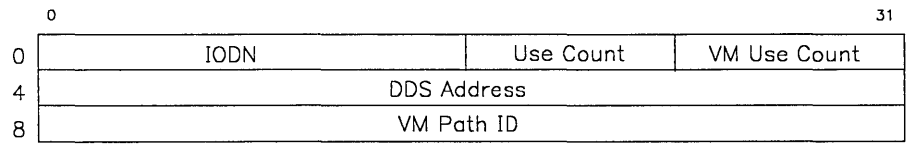


Figure 4-2. Device Manager IODN Table

The fields in the preceding figure are defined as follows:

- IODN — indicates the IODN of the port or device. A value of zero indicates an unused entry.
- Use Count — indicates the total number of times the IODN has been opened. This value is increased by one for each **open** of the IODN and decreased by one for each **close** of the IODN. If the count reaches zero, the device manager detaches from the IODN and marks the table entry as unused.
- VM Use Count — indicates the total number of times the IODN has been opened by virtual machine LLC processes. This value is increased and decreased the same way as the Use Count. If this value reaches zero, the virtual machine-to-device driver path is detached. Because all LLCs from a virtual machine share the same path to the device driver, the device manager uses this value to determine when to delete the path.
- DDS Address — indicates the location of the DDS associated with the IODN.
- VM Path ID — contains the ID of the virtual machine-to-device manager path (if the IODN has been opened by a virtual machine LLC).

- **_detchq**

The device manager uses the **_detchq** routine when it receives the last **close** command for a given IODN.

The first task the manager performs is to detach the LLC process from the device driver. The parameter to this call is defined as follows:

- Path ID — ID of the LLC-to-device driver path.

When the LLC is detached, the device manager then detaches from the driver's IODN. Because this is the last detach from the device driver, the driver's Terminate function will also be called. The parameter to this call is:

- Path ID — ID of the device manager-to-device driver path.

- **_enque**

After the device manager establishes the appropriate paths with **_attchq**, the device manager can send commands to the driver with the **_enque** routine and Send Command queue elements. The commands available to this interface are **open** and **IPL adapter**. All block I/O device drivers must support the **open**, but only those drivers for adapters that require a microcode load must support **IPL adapter**. In addition, acknowledgment queue elements are required by the device manager for both commands.

The commands are defined as follows:

- **open**

  The **open** command is the first command passed to the device driver from the manager. This command is used primarily to allow the driver to provide its DDS address to the manager. At this time the device driver can also initialize its data structures and adapter.

  The following figure shows the Send Command queue element used for the **open** command.

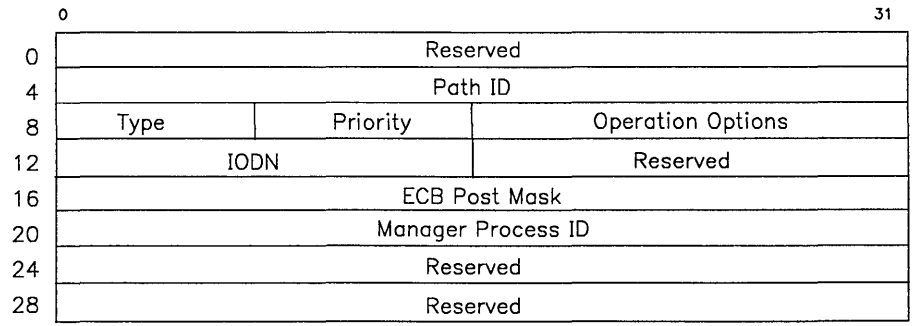


Figure 4-3. Manager-to-Driver Send Command Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — indicates the ID of the device manager-to-device driver path
- Type — set to 1 for Send Command queue element
- Priority — always set to zero
- Operation Options:

  The device option field for an **open** command is 2. The Command Extension bit is set to zero; the Interrupt on Completion and Interrupt on Error bits are set to one.

- IODN — indicates the IODN of the adapter port
- ECB Mask — indicates the ECB mask which will be used by the driver when it performs a **_post** to request additional buffers.
- Manager Process ID — indicates the manager's PID used by the driver when doing a **_post** to request additional buffers.

The acknowledgment queue element for an **open** request is shown in the following figure.

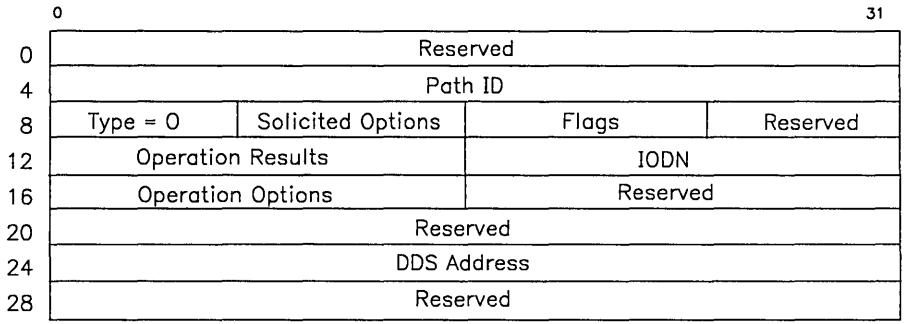


Figure 4-4. Manager-to-Driver Acknowledgment Queue Element

The fields in the preceding figure are defined as follows:
- Path ID — indicates the ID of the device manager-to-device driver path
- Type — set to 0 for acknowledgment queue element
- Solicited Options — contains the high-order byte of the Operation Options field
- Flags — always set to 0x14, solicited acknowledgment
- Operation Results — see "Operation Results" on page 4-42
- IODN — indicates the IODN of the adapter port
- Operation Options — contains the Operation Options field from the Send Command queue element
- DDS Address — indicates the address of the adapter port's DDS.

- **IPL adapter**

The **IPL adapter** command is used to pass adapter microcode to the VRM device driver. The device driver returns operation results of 0x01 when opened if it requires microcode to IPL the adapter.

The Send Command queue element for **IPL adapter** requires a command extension. The data in the command extension is the microcode required by the adapter.

Figure 4-5 on page 4-10 shows the Send Command queue element used for the **IPL adapter** command.

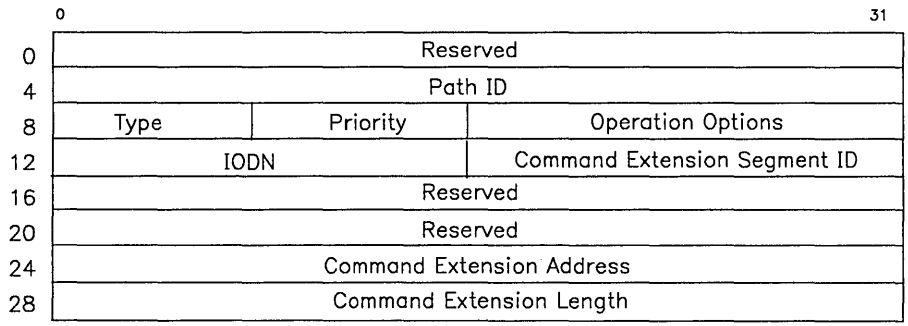


Figure 4-5. IPL Adapter Send Command Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — indicates the ID of the device manager-to-device driver path
- Type — set to 1 for Send Command queue element
- Priority — always set to zero
- Operation Options:

  The device option field is set to 5 for the **IPL adapter** command. The Command Extension bit is set to one; the Interrupt on Completion and Interrupt on Error bits are set to one.
- IODN — indicates the IODN of the adapter port
- Command Extension Seg. ID — indicates the segment ID of the buffer that contains the adapter microcode
- Command Extension Address — indicates the address of the buffer that contains the adapter microcode
- Command Extension Length — indicates the length of the buffer that contains the adapter microcode.

The acknowledgment queue element for the **IPL adapter** request is shown in the figure on the following page.

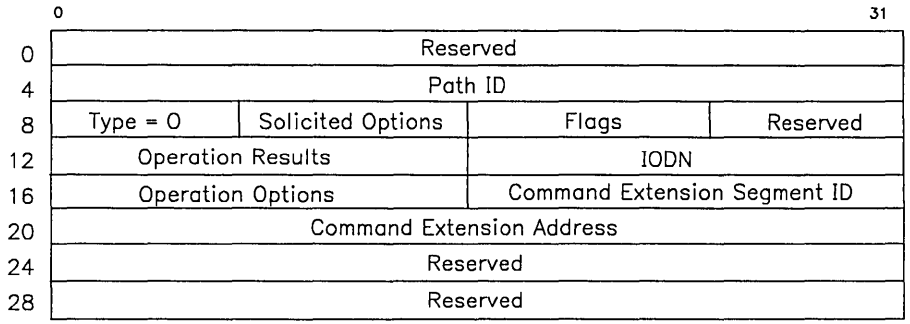


Figure 4-6. IPL Adapter Acknowledgment Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — indicates the ID of the device manager-to-device driver path
- Type — set to 0 for acknowledgment queue element
- Solicited Options — contains the high-order byte of the Operation Options field
- Flags — always set to 0x14, solicited acknowledgment
- Operation Results — see "Operation Results" on page 4-42
- IODN — indicates the IODN of the adapter port
- Operation Options — contains the Operation Options field from the input Send Command queue element field is reserved for acknowledgment to a **close** request.
- Command Extension Seg. ID — indicates the segment ID of the buffer that contains the adapter microcode
- Command Extension Address — indicates the address of the buffer that contains the adapter microcode.

For adapters that require a microcode load, the following conventions must be followed:

- The configuration process must install the microcode in the system as a code module. The IOCN assigned to this module must be placed in each DDS associated with a port on the adapter.
- The device driver for adapters that require a microcode load must respond to the first **open** command with operation results of 0x0001 to indicate to the device manager that the microcode is required.
- The device manager then obtains the microcode IOCN from the port's DDS. The manager determines the virtual address of the code and issues the **IPL adapter** command. The command extension area contains the microcode.
- The adapter microcode must have an export symbol associated with it so the device manager can determine the code's virtual address. This export symbol must be "IPLCODE" or "_IPLCODE" for all adapters that require a microcode load. The first word of the code module must indicate the length in bytes of the microcode (including the length word itself). The actual microcode can be of any length, but must begin on a fullword boundary.

- **_post**

  The block I/O device driver uses the **_post** routine to notify the device manager that additional pinned receive buffers are required. The parameters consist of the device manager's process ID and the assigned ECB mask passed to the driver with the **open** command. The ECB mask identifies the device that needs the additional pinned buffers and is converted to an index into the device manager's IODN table (see Figure 4-2 on page 4-7). The device manager does not notify the device driver when the requested buffers are allocated.

  The device manager uses **_wait** to be notified of the **_post** from the device driver.

LLC Process-to-Device Driver Interface

After the device manager sets up the paths, allocates the buffers, and performs the rest of its tasks, the LLC process can send commands directly to the device driver. The following commands can be used:

- **Start device**

  This command notifies the device driver of a network ID that it can use. After a device driver receives this command, data can be sent and received for the specified network ID. Operating system LLC processes use the **Send Command SVC** to issue this command. VRM LLC processes use the **_enque** routine with Send Command queue elements to issue this command.

- **Write**

  This command is used to send data. Three forms of this command are available. Depending on the amount of data you want to send, use one of the following forms of **write**:

  - **Write short**

    This form of the write command allows you to pass up to 8 bytes of data to the device driver without using a buffer. This form uses the **Send Command SVC** (for operating system LLCs) or **_enque** routines (for VRM LLCs) to send the data.

  - **Write long**

    This form allows you to send a single buffer of data to the device driver. The maximum amount of data the buffer can contain is device-dependent. This form uses the **Send Command SVC** to send the data buffer from an operating system LLC and the **_enque** routine to send data from VRM LLCs.

  - **Start I/O write**

    This form of the write command should be used only to send multiple buffers of data to the device driver with a single command. This form of the write command uses the **Start I/O SVC** for operating system LLCs and the Start I/O queue element of the **_enque** routine for VRM LLCs.

- **Halt device**

  This command notifies the device driver that a network ID is no longer needed. When this command is complete, data cannot be sent or received for the specified network ID. The halt device command is issued with a **Send Command SVC** for operating system LLCs and the **_enque** routine for VRM LLCs.

- Device-dependent commands

  In addition to the commands defined above, other commands can be defined by cooperating LLC processes and device drivers. The **Send Command SVC** (for operating system LLCs) and the Send Command queue element of the **_enque** routine (for VRM LLCs) allow you to define any device-specific operation that you require.

The following sections describe each command in more detail.

Start Device

The **Start device** command establishes a session between an LLC process and the device driver for a specific device. A network ID determines to which LLC the driver passes receive data. The driver must ensure that the ID is not already in use and that the number of IDs supported is not exceeded.

A driver can route all received data to a single LLC by specifying zero for the network ID length field in the driver's DDS. In this case, the driver does not receive a network ID and the command extension to the **Start device** consists of only the first word of Figure 4-8 on page 4-14.

The driver must ensure that an LLC is not notified of received data before the LLC's **Start device** is complete. Before an LLC can receive data, it must obtain the device ring queue address, which is returned in the **Start device** acknowledgment queue element.

Because the device driver correlator is also returned in the **Start device** acknowledgment queue element, this command should not be issued synchronously (the synchronous operation bit in the operation options field should be set to zero).

For VRM LLCs, this command is issued with the **_enque** routine's Send Command queue element. The following figure shows the queue element for this command.

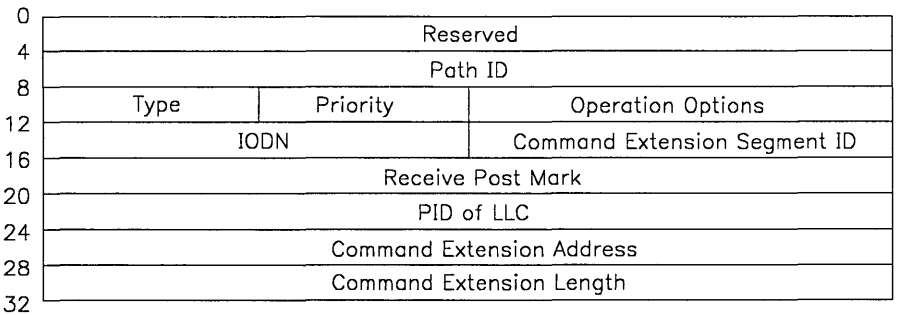


Figure 4-7. Start Device Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — ID of the LLC-to-device driver path.
- Type — set to 1 for Send Command queue element.
- Priority — set to zero.
- Operation Options — command extension bit must be 1. Device option equals 7. For other values, see "Operation Options" on page 4-26.
- IODN — of the port or device.
- Command Extension Seg ID — segment ID of the command extension area.
- Receive POST Mask — for VRM LLCs only, indicates the bit to use to inform the LLC of received data. For OS LLCs, this field is set to zero.
- PID of LLC — for VRM LLCs only, indicates the process ID of the LLC. For OS LLCs, this field is set to zero.
- Command Extension Address — address of the command extension area.
- Command Extension Length — length of the command extension area. This area must be large enough to hold both input and returned parameters.

The command extension is required to pass additional parameters. The format of the command extension is shown below.

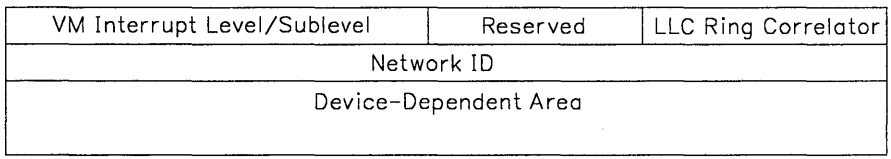


Figure 4-8. Start Device Command Extension. Note that the command extension area also returns data to the LLC process. Therefore, the area must be large enough to pass parameters to and from the device driver.

The fields in the preceding figure are defined as follows:

- VM Interrupt Level/Sublevel — for OS LLCs only, indicates the level and sublevel on which to return virtual interrupts. For VRM LLCs, this field must be set to zero.
- LLC Ring Correlator — a value assigned by the LLC to acknowledgment queue elements or virtual interrupts returned for the indicated network ID.
- Network ID — specifies an LLC process on the network. The length of this field is specified in the device's DDS.
- Device-Dependent Area — used to pass additional device-dependent data, if any.

The figure on the following page shows a returned queue element for this command.

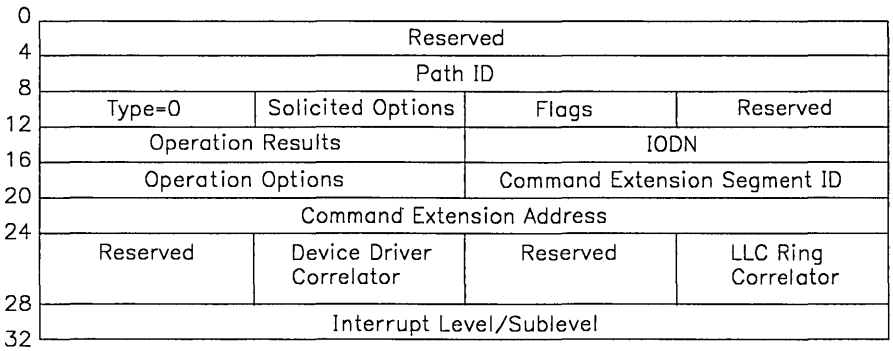


Figure 4-9. Start Device Acknowledgment Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — ID of the LLC-to-device driver path.
- Type — equals zero for an acknowledgment queue element.
- Solicited Options — contains the high-order byte of the operation options field.
- Flags — set to 0x14 to indicate a solicited interrupt.
- Operation Results — see "Operation Results" on page 4-36.
- IODN — of the device or port.
- Operation Options — operation options from the input queue element.
- Command Extension Seg ID — segment ID of the command extension area.
- Command Extension Address — address of the command extension area.
- Device Driver Correlator — correlator value assigned by the driver.
- LLC Ring Correlator — value supplied in the input queue element.
- Interrupt Level/Sublevel — for OS LLCs only, indicates the level and sublevel on which to return virtual interrupts. For VRM LLCs, this field is set to zero.

The returned command extension area for this command (if any) consists of a 32-bit device queue ring address. This address is assigned to the network ID that was passed with the **Start device** command.

After the ring queue address is a variable-length field of device-dependent data that consists of the portions of the command extension area used to pass the network ID and device-dependent data to the driver.

For OS LLCs, this command is issued with a **Send Command SVC**. Input parameters are defined as follows:

- GPR2 = device driver IODN (bits 0-15), operation options (bits 16-31). The device option for a **Start** command is 7. See "Operation Options" on page 4-26 for other values.
- GPR3,4 = reserved.
- GPR5 = command extension address.
- GPR6 = command extension length.
- GPR7 = ID of the device driver-to-LLC path.

OS LLCs also use the command extension format shown in Figure 4-8 on page 4-14.

Returned information contained in the Send Command PSB includes:

- Status flags — set to 0x14 for a solicited interrupt.
- Overrun count — set to zero.
- Operation result — for operation results (bits 0-15) see "Operation Results" on page 4-36; device driver IODN (bits 16-31).
- Data word 1 — operation options (bits 0-15); segment ID of the command extension (bits 16-31).
- Data word 2 — command extension address.
- Data word 3 — device driver correlator (bits 8-15); LLC ring correlator (bits 24-31); all other bits reserved.

The returned command extension area is the same as that described for a VRM LLC.

Halt Device

The **Halt device** command is used by the LLC process to end a session with the device driver. The device driver correlator is passed as an input parameter so the driver knows which network ID to terminate.

The following figure shows an input queue element for this command.

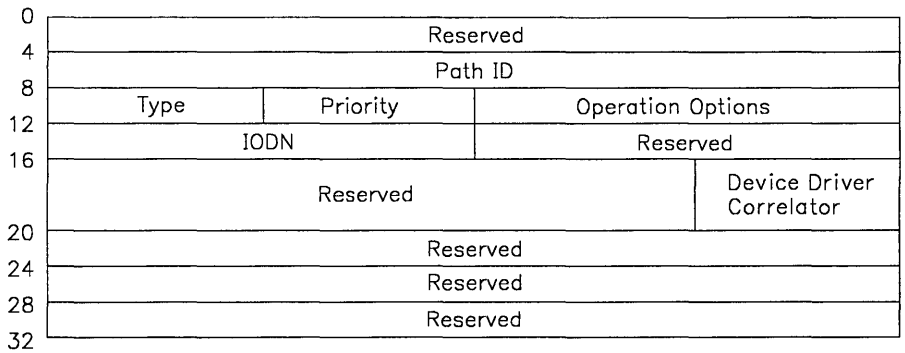


Figure 4-10. Halt Device Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — ID of the LLC-to-device driver path.
- Type — set to 1 for Send Command queue element.
- Priority — set to zero.
- Operation Options — command extension bit must be 1. Device option equals 8. For other values, see "Operation Options" on page 4-26.
- IODN — of the port or device.
- Device Driver Correlator — correlator value that was returned to the LLC process upon completion of the **Start device** command.

The following figure shows a returned queue element for this command.

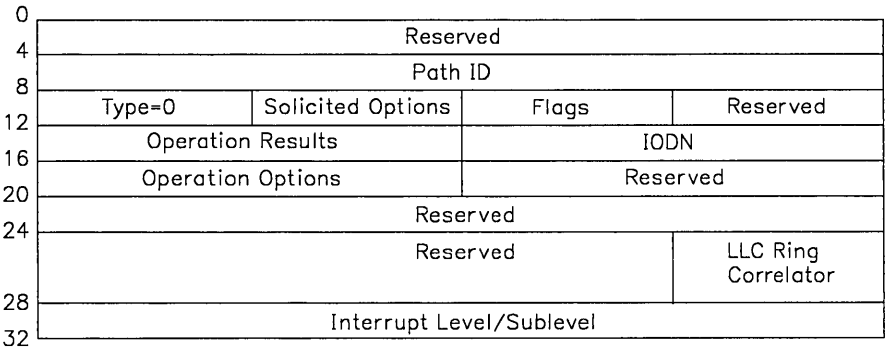


Figure 4-11. Halt Device Acknowledgment Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — ID of the LLC-to-device driver path.
- Type — equals zero for an acknowledgment queue element.
- Solicited Options — contains the high-order byte of the operation options field.
- Flags — set to 0x14 to indicate a solicited acknowledgment.
- Operation Results — see "Operation Results" on page 4-36.
- IODN — of the device or port.
- Operation Options — operation options from the input queue element.
- LLC Ring Correlator — correlator value that was supplied with the **Start device** command.
- Interrupt Level/Sublevel — for OS LLCs only, indicates the level and sublevel on which to return virtual interrupts. For VRM LLCs, this field is set to zero.

For OS LLCs, this command is issued with a **Send Command SVC**. Input parameters are defined as follows:

- GPR2 = device driver IODN (bits 0-15), operation options (bits 16-31). The device option for the **halt** command is 8. See "Operation Options" on page 4-26 for other values.
- GPR3 = device driver correlator (bits 24-31).
- GPR4,5,6 = reserved.
- GPR7 = ID of the device driver-to-LLC path.

Returned information contained in the Send Command PSB includes:

- Status flags — set to 0x14 for a solicited interrupt.
- Overrun count — set to zero.
- Operation result — for operation results (bits 0-15) see "Operation Results" on page 4-36; device driver IODN (bits 16-31).
- Data word 1 — operation options (bits 0-15); reserved (bits 16-31).
- Data word 2 — reserved.

- Data word 3 — reserved (bits 0-23); LLC ring correlator that corresponds to the deleted network ID (bits 24-31).

Write Long

The **Write long** command passes a buffer of data to a block I/O device driver for transmission on a local-area network or communications line. This command uses Send Command elements and command extensions. The command extension area consists of the data buffer to be sent. An LLC can use a buffer obtained from the buffer pool or from its own memory to contain the data.

The buffer format bit described in "Operation Options" on page 4-26 indicates the format of the buffer. When the buffer format bit is set, the buffer contains only data. When the buffer format bit is zero, the buffer is of the format shown in Figure 4-27 on page 4-36.

If the buffer is from the buffer pool and the entire buffer is passed to the driver, the LLC can request that the buffer be returned to the pool by setting the free buffer bit (also described in "Operation Options" on page 4-26).

Acknowledgment to this call indicates only that the data was sent and not that it was received on the other end. Therefore, to reduce system overhead, you can set the interrupt on completion bit of the operation options field to zero.

The following figure shows an input queue element for this command.

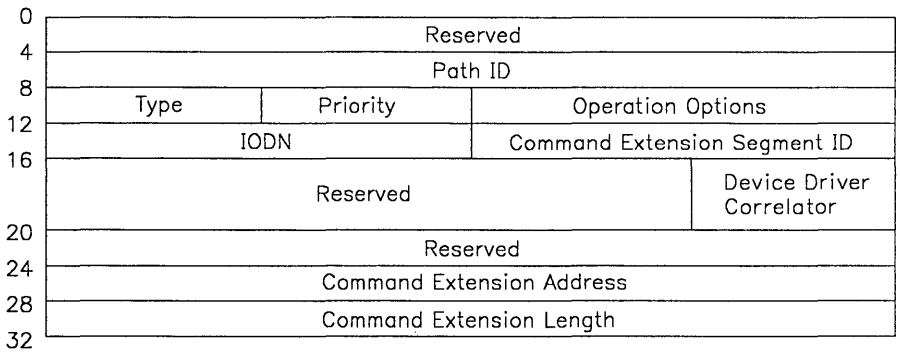


Figure 4-12. Write Long Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — ID of the LLC-to-device driver path.
- Type — set to 1 for Send Command queue element.
- Priority — set to zero.
- Operation Options — command extension bit must be 1. Device option equals 6. For other values, see "Operation Options" on page 4-26.
- IODN — of the device driver.
- Command Extension Segment ID — segment ID of the command extension area.

- Device Driver Correlator — the correlator value that was returned to the LLC from the **Start device** command.
- Command Extension Address — address of the data buffer.
- Command Extension Length — length of the data buffer.

The following figure shows a returned queue element for this command.

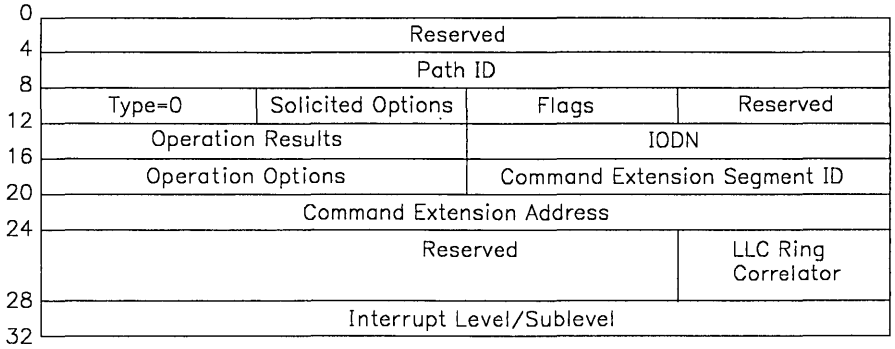


Figure 4-13. Write Long Acknowledgment Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — ID of the LLC-to-device driver path.
- Type — equals zero for an acknowledgment queue element.
- Solicited Options — contains the high-order byte of the operation options field.
- Flags — set to 0x14 to indicate a solicited acknowledgment.
- Operation Results — see "Operation Results" on page 4-36.
- IODN — of the device driver.
- Operation Options — operation options from the input queue element.
- Command Extension Segment ID — segment ID of the data buffer.
- Command Extension Address — address of the data buffer.
- LLC Ring Correlator — correlator value that was supplied with the **Start device** command.
- Interrupt Level/Sublevel — for OS LLCs only, indicates the level and sublevel on which to return virtual interrupts. For VRM LLCs, this field is set to zero.

For OS LLCs, this command is issued with a **Send Command SVC**. Input parameters are defined as follows:

- GPR2 = device driver IODN (bits 0-15), operation options (bits 16-31). The device option for the **Write long** command is 6. See "Operation Options" on page 4-26 for other values.
- GPR3 = device driver correlator (bits 24-31).
- GPR4 = reserved.
- GPR5 = command extension address.
- GPR6 = command extension length.
- GPR7 = ID of the device driver-to-LLC path.

Returned information contained in the Send Command PSB includes:

- Status flags — set to 0x14 for a solicited interrupt.
- Overrun count — set to zero.
- Operation result — for operation results (bits 0-15) see "Operation Results" on page 4-36; device driver IODN (bits 16-31).
- Data word 1 — operation options (bits 0-15); reserved (bits 16-31).
- Data word 2 — command extension address.
- Data word 3 — reserved (bits 0-23); LLC ring correlator (bits 24-31).

Write Short

The **Write short** command is similar to the **Write long** command, but where **Write long** uses a buffer to send data, the **Write short** sends data within the Send Command queue element itself. This restricts the **Write short** command to sending only eight bytes of data per queue element.

The **Write short** command has no command extension, so the command extension bit in the operation options field is set to zero. The buffer format and free buffer fields in the operation options area do not apply to this command.

The acknowledgment to this command indicates only that the data was sent, not that the data was received at the other end. Therefore, to reduce system overhead, you can set the interrupt on completion bit of the operation options field to zero.

The following figure shows an input queue element for this command.

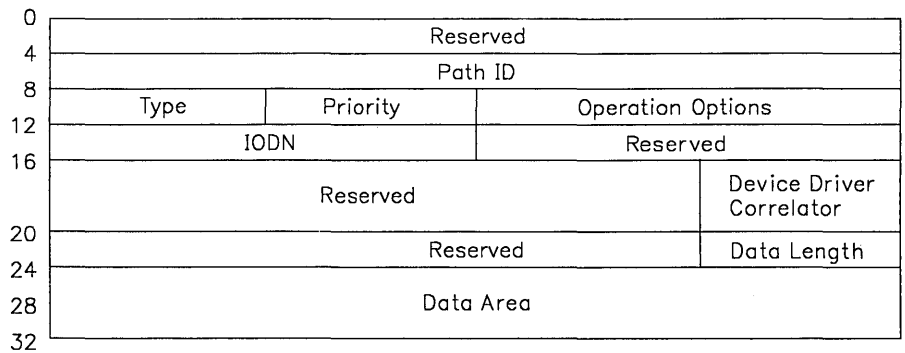


Figure 4-14. Write Short Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — ID of the LLC-to-device driver path.
- Type — set to 1 for Send Command queue element.
- Priority — set to zero.
- Operation Options — command extension bit must be 0. Device option equals 6. For other values, see "Operation Options" on page 4-26.

- IODN — of the device driver.
- Device Driver Correlator — the correlator value that was returned to the LLC from the **Start device** command.
- Data Length — number of data bytes (up to 8) in the data area.
- Data Area — contains the data to be sent. If fewer than 8 bytes are being passed, the data must be left-justified within the data area.

The acknowledgment queue element for a **Write short** request has the same format as the acknowledgment queue element for a **Write long** request, except that the command extension segment ID and command extension address fields defined for the **Write long** acknowledgment are reserved (set equal to zero) for the **Write short** acknowledgment.

For OS LLCs, this command is issued with a **Send Command SVC**. Input parameters are defined as follows:

- GPR2 = device driver IODN (bits 0-15), operation options (bits 16-31). The device option for the **Write long** command is 6. See "Operation Options" on page 4-26 for other values.
- GPR3 = device driver correlator (bits 24-31).
- GPR4 = length of data (bits 24-31).
- GPR5 = data bytes 1 through 4, left-justified
- GPR6 = data bytes 5 through 8, left-justified.
- GPR7 = ID of the device driver-to-LLC path.

Returned information contained in the Send Command PSB includes:

- Status flags — set to 0x14 for a solicited interrupt.
- Overrun count — set to zero.
- Operation result — for operation results (bits 0-15) see "Operation Results" on page 4-36; device driver IODN (bits 16-31).
- Data word 1 — operation options (bits 0-15); reserved (bits 16-31).
- Data word 2 — reserved.
- Data word 3 — reserved (bits 0-23); LLC ring correlator (bits 24-31).

Write (Start I/O)

For sending multiple data buffers, the **Write** command can be issued with Start I/O queue elements (VRM LLCs) or the **Start I/O SVC** (OS LLCs). A command control block associated with Start I/O requests consists of a command header and one or more command elements. Each command element defines a data buffer to be sent. The LLC can use buffers from the buffer pool or from its own memory.

The buffer format bit described in "Operation Options" on page 4-26 indicates the format of the buffer. When the buffer format bit is set, the buffer contains only data. When the buffer format bit is zero, the buffer is of the format shown in Figure 4-27 on page 4-36.

If the buffer is from the buffer pool and the entire buffer is passed to the driver, the LLC can request that the buffer be returned to the pool by setting the free buffer bit (also described in "Operation Options" on page 4-26).

Acknowledgment to this call indicates only that the data was sent and not that it was received on the other end. Therefore, to reduce system overhead, you can set the interrupt on completion bit of the operation options field to zero.

The following figure shows an input queue element for this command.

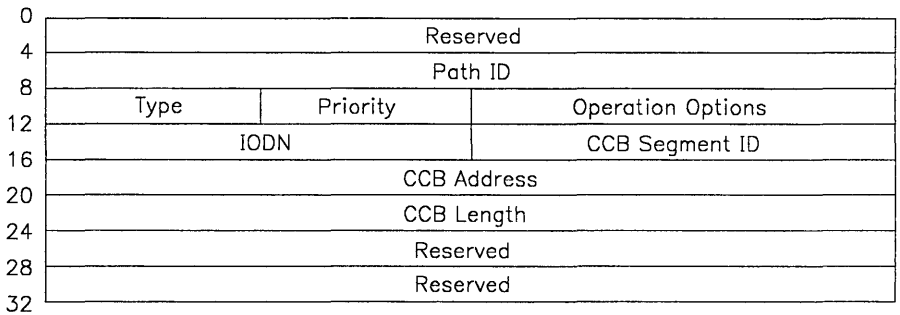


Figure 4-15. Write (Start I/O) Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — ID of the LLC-to-device driver path.
- Type — set to 2 for Start I/O queue element.
- Priority — set to zero.
- Operation Options — command extension bit must be 1. Device option equals 1. For other values, see "Operation Options" on page 4-26.
- IODN — of the device driver.
- CCB Segment ID — segment ID of the command control block.
- CCB Address — address of the CCB.
- CCB Length — length of the CCB.

The CCB header and elements are described in the **Start I/O SVC** definition in *VRM Programming Reference*. Fields defined for use in the block I/O subsystem are shown below:

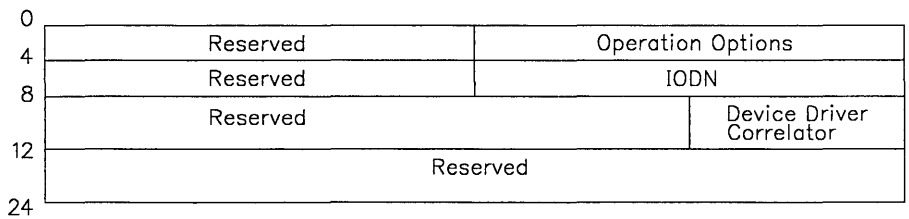


Figure 4-16. CCB Header for Write (Start I/O)

The fields in the preceding figure are defined as follows:

- Operation Options — command element bit must be set, device option field equals 1 (for write). For other values, see "Operation Options" on page 4-26.
- IODN — of the device driver.
- Device Driver Correlator — correlator value that was returned to the LLC from the **Start device** command.

The CCB is followed by one or more command elements (also shown in *VRM Programming Reference*). For block I/O subsystem Start I/O requests, the command element fields are defined as follows:

- Word 1 —
  Bit 15 (link bit) is set to one to indicate additional command elements to follow. For the last command element in the chain, this bit is set to zero.
  Bits 16-31 contain the segment ID of the data buffer.
- Word 2 — contains the length of the data buffer, in bytes.
- Word 3 — contains the address of the data buffer.

The following figure shows a returned queue element for this command.

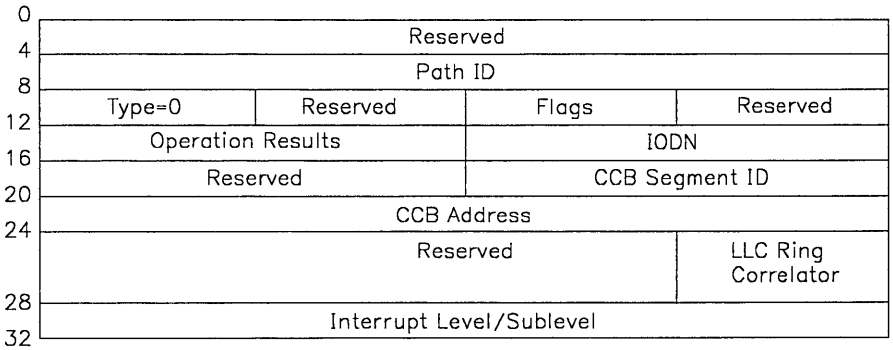


Figure 4-17. Write (Start I/O) Acknowledgment Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — ID of the LLC-to-device driver path.
- Type — equals zero for an acknowledgment queue element.
- Flags — set to 0x24 to indicate a solicited acknowledgment.
- Operation Results — see "Operation Results" on page 4-36.
- IODN — of the device driver.
- CCB Segment ID — segment ID of the CCB.
- CCB Address — address of the CCB.
- LLC Ring Correlator — correlator value that was supplied with the **Start device** command.
- Interrupt Level/Sublevel — for OS LLCs only, indicates the level and sublevel on which to return virtual interrupts. For VRM LLCs, this field is set to zero.

For OS LLCs, this command is issued with a **Start I/O SVC**. Input parameters are defined as follows:

- GPR2 = CCB address.
- GPR3 = ID of the device driver-to-LLC process path.

The CCB and command elements have the same format as those defined for VRM LLCs, with one exception: the command element field in bits 16-31 of word 1 (data buffer segment ID) is reserved (set equal to zero) for OS LLCs.

Returned information contained in the Start I/O PSB includes:

- Status flags — set to 0x24 for a solicited interrupt.
- Overrun count — set to zero.
- Operation result — for operation results (bits 0-15) see "Operation Results" on page 4-36; device driver IODN (bits 16-31).
- Data word 1 — reserved (bits 0-15); CCB segment ID (bits 16-31).
- Data word 2 — CCB address.
- Data word 3 — reserved (bits 0-23); LLC ring correlator (bits 24-31).

Device-Dependent Commands

The block I/O subsystem allows cooperating components to define their own commands. For VRM LLCs, these device-dependent commands use the **_enque** routine with Send Command queue elements.

The following figure shows an input queue element for a device-dependent command.

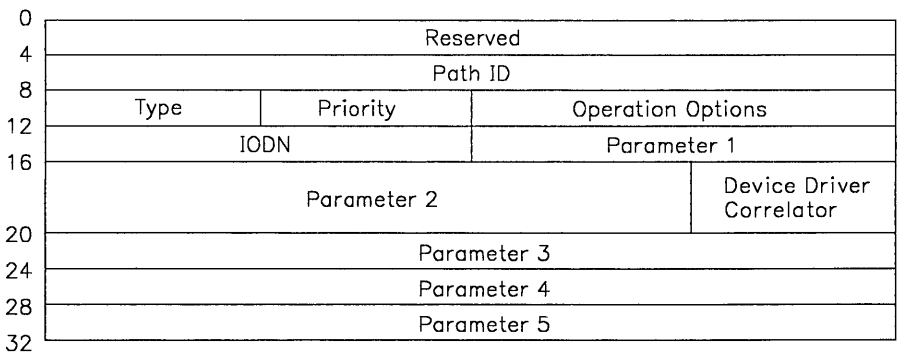


Figure 4-18. Device-Dependent Command Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — ID of the LLC-to-device driver path.
- Type — set to 1 for Send Command queue element.
- Priority — set to zero.

- Operation Options — vary depending on the command. See "Operation Options" on page 4-26 for possible values for this field.
- IODN — of the device driver.
- Parameter 1 — if a command extension is specified in the operation options field, this field contains the segment ID of the command extension. Otherwise, this is a device-dependent parameter.
- Parameter 2 — is a device-dependent parameter.
- Device driver Correlator — is the correlator value returned to the LLC from the **Start Device** command.
- Parameter 3 — is a device-dependent parameter.
- Parameter 4 — if a command extension is specified in the operation options field, this field contains the address of the command extension. Otherwise, this is a device-dependent parameter.
- Parameter 5 — if a command extension is specified in the operation options field, this field contains the length of the command extension. Otherwise, this is a device-dependent parameter.

The following figure shows a returned queue element for a device-dependent command.

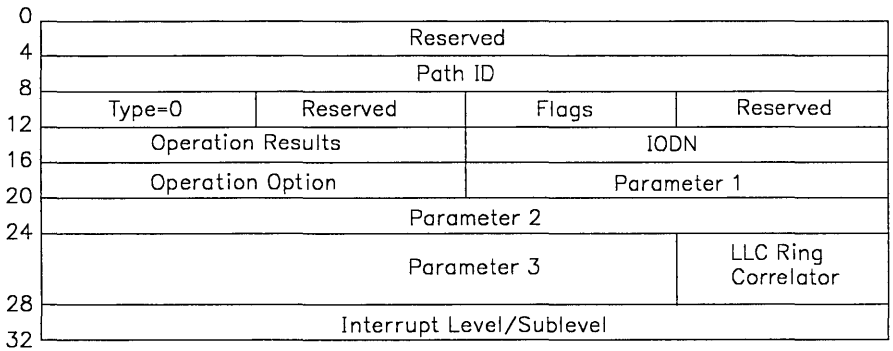


Figure 4-19. Device-Dependent Acknowledgment Queue Element

The fields in the preceding figure are defined as follows:

- Path ID — ID of the LLC-to-device driver path.
- Type — equals zero for an acknowledgment queue element.
- Solicited Options — contains the high-order byte of the operation options field.
- Flags — set to 0x14 to indicate a solicited acknowledgment.
- Operation Results — see "Operation Results" on page 4-36.
- IODN — of the device driver.
- Operation Options — operation options from the input queue element.
- Parameter 1 — if a command extension is specified in the operation options field, this field contains the segment ID of the command extension. Otherwise, this is a device-dependent value.
- Parameter 2 — if a command extension is specified in the operation options field, this field contains the address of the command extension. Otherwise, this is a device-dependent value.
- Parameter 3 — is a device-dependent value.
- LLC Ring Correlator — correlator value that was supplied with the **Start device** command.

- Interrupt Level/Sublevel — for OS LLCs only, indicates the level and sublevel on which to return virtual interrupts. For VRM LLCs, this field is set to zero.

Device-dependent commands can also be defined for OS LLCs. OS LLCs use the **Send Command SVC** to issue device-dependent commands. Input parameters to the SVC are defined as follows:

- GPR2 = device driver IODN (bits 0-15); operation options (bits 16-31). The possible values for this field are defined in "Operation Options."
- GPR3 = device-dependent value (bits 0-23); device driver correlator (bits 24-31).
- GPR4 = is a device-dependent parameter.
- GPR5 = if a command extension is specified in the operation options field, this register contains the address of the command extension area. Otherwise, GPR5 contains a device-dependent parameter.
- GPR6 = if a command extension is specified in the operation options field, this register contains the length of the command extension area. Otherwise, GPR6 contains a device-dependent parameter.
- GPR7 = ID of the device driver-to-LLC path.

Returned information contained in the Send Command PSB includes:

- Status flags — set to 0x14 for a solicited interrupt.
- Overrun count — set to zero.
- Operation result — for operation results (bits 0-15) see "Operation Results" on page 4-36; device driver IODN (bits 16-31).
- Data word 1 — operation options (bits 0-15); if a command extension was specified in the operation options field, bits 16-31 contain the segment ID of the extension. Otherwise, bits 16-31 are device-dependent.
- Data word 2 — if a command extension was specified in the operation options field, this word contains the address of the extension. Otherwise, this field is device-dependent.
- Data word 3 — device-dependent value (bits 0-23); LLC ring correlator (bits 24-31).

Operation Options

Operation options fields determine how command processing is performed. The **Send Command SVC** indicates its operation options in GPR2. **Start I/O SVC** indicates its operation options in a command control block. The Send Command and Start I/O queue elements for the **_enque** routine also contain operation options fields. For block I/O requests, this field is defined as shown in the figure on the next page.

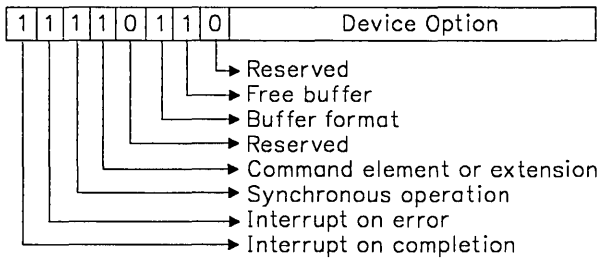


Figure 4-20. Block I/O Operation Options Field

The bits in the preceding figure are defined as follows:

- Interrupt on completion — when set, causes the operating system to be interrupted upon completion of the I/O operation
- Interrupt on error — when set, causes the operating system to be interrupted if the I/O operation is terminated
- Synchronous operation — when set, places the current operating system in a wait state pending the completion of the I/O operation. If this bit is set, GPR2 contains the operation results of the block I/O request initiated with **_enque, Send Command SVC, or Start I/O SVC**. If this bit is not set, GPR2 contains only the return code from the **_enque, Send Command SVC, or Start I/O SVC**.
- Command element or extension
  - Command element — when set, indicates that the command control block associated with a Start I/O request has one or more command elements
  - Command extension — when set, indicates that the Send Command queue element associated with this I/O request has a command extension field.
- Buffer format — indicates the format of the data buffer for **Write** commands. When set, this bit indicates that the entire command extension or command element is to be treated as data and sent. When this bit is zero, the buffer area is of the format shown in Figure 4-27 on page 4-36.
- Free buffer — when set, indicates that the device driver should return the data buffer to the buffer pool when a **Write** request is completed. This applies only if the buffer was obtained from the buffer pool and only if the Buffer Format bit is not set.
- Device option — indicates the operation to be performed. For a **Start I/O SVC**, the only valid block I/O subsystem value is 1 (for a **Write** request). For a **Send Command SVC**, the following values are valid:
  0, 1, 4 = reserved
  2 = open
  3 = close
  5 = IPL adapter
  6 = write (both short and long forms)
  7 = start device
  8 = halt device
  > 8 = device-dependent.

Receive Data Mechanism

The block I/O subsystem does not define a **Read** or **Receive** command, but a **Read** or **Receive** is implied whenever a **Start Device** is completed.

Received data is automatically routed to the LLC process by way of the pinned receive buffers that the device manager provides to the block I/O device driver. When a device driver processes an adapter interrupt for received data, the driver gets a buffer from its SLIH ring queue, takes the data from the adapter and places it into the buffer. By comparing the network ID included in the received data to the IDs in the network correlation table, the driver determines to which LLC process to send the data. The driver then places the buffer's segment offset into the appropriate LLC's device ring queue.

If the device ring queue was empty, the LLC process is informed (by way of a **_post** for VRM LLCs or a virtual interrupt for OS LLCs) that a buffer of data has been received. If the device ring queue was not empty, no action is taken because the LLC is expected to empty the device ring queue when it has been informed that buffers are available.

The amount of data received is indicated in the data length field of the receive buffer. The LLC uses the data offset field to determine the actual start of the data. Figure 4-27 on page 4-36 shows the location of these fields.

The driver increments a 'buffers used' count each time it removes a buffer from the SLIH ring queue. The result is compared to the SLIH threshold value contained in its DDS. If the values are the same, the driver issues a **_post** to the device manager, indicating the need for additional pinned receive buffers.

If the SLIH ring queue is empty when a receive buffer is needed, or the device ring queue is full when a receive buffer is to be added to it, an unsolicited queue element or virtual interrupt is generated to indicate the error.

If the receive buffer has overflowed, the low-order bit of the flag field should be set. The remaining data is discarded.

VRM Block I/O Device Driver DDS

The header and hardware characteristics sections of a block I/O device driver follow the formats shown in Figure 2-2 on page 2-5 and Figure 2-3 on page 2-6, respectively. In order for a device driver to work properly in the block I/O subsystem, the device characteristics section must follow the format shown in Figure 4-21.

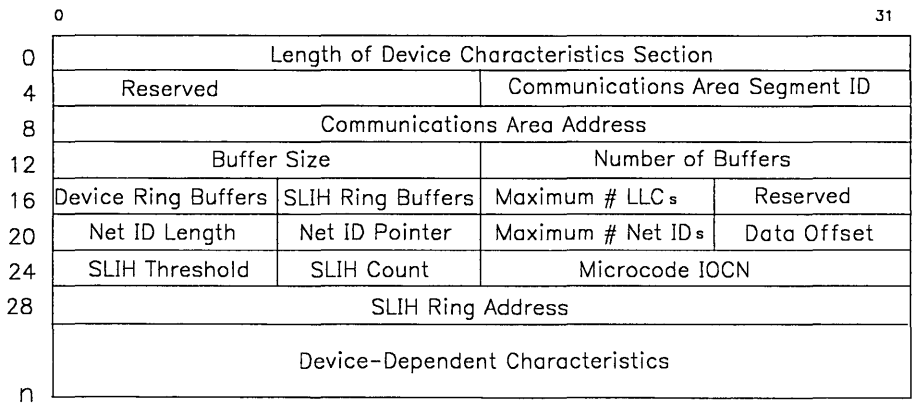


Figure 4-21. Device Characteristics – Block I/O Device Driver

The fields in the preceding figure are defined as follows:

- Length of Device Characteristics Section – is the length in words of this section.
- Communications Area Segment ID – is the ID of the segment where the block I/O communications area resides.
- Communications Area Address – indicates the segment offset of the block I/O communications area.
- Buffer Size – indicates the length of each individual buffer in the buffer pool. This value must be a multiple of 4 so that buffers hold fullwords of data.
- Number of Buffers – indicates the number of defined buffers.
- Device Ring Buffers – is the number of buffers placed on each device ring queue.
- SLIH Ring Buffers – is the number of receive buffers placed on the SLIH ring queue.
- Maximum # LLCs – indicates how many logical link controls can attach to the device at any one time.
- NetID Length – is the length of the network ID in bytes.
- NetID Pointer – indicates the location of the network ID in the receive data. This value is the offset in bytes from the start of the receive data.
- Maximum # NetIDs – indicates how many network IDs the device can support.
- Data Offset – indicates an offset (in bytes) into the data area of a receive buffer at which the device driver is to place received data. This field is not used by all device drivers.

- SLIH Threshold – is the number of SLIH ring queue buffers the device driver can use before it must request additional buffers from the block I/O device manager.
- SLIH Count – indicates how many SLIH ring queue buffers the device driver has moved to receive ring queues. The driver initializes this field to zero and increments it each time it moves a buffer to a receive ring queue. If the SLIH count is equal to the SLIH threshold, the driver resets the count to zero and issues a **_post** call to the device manager to request additional buffers.
- Microcode IOCN – is the IOCN of the IPL code for those adapters that require a separate IPL. For adapters that do not require an IPL, this field is zero.
- SLIH Ring Address – is the address of the SLIH ring. This value is filled in by the device driver when it receives an **open** request from the block I/O device manager.
- Device-Dependent Characteristics – is an optional area that can be defined by each individual device.

Network ID Correlation Table

A network ID is a variable length value used to differentiate between LLC processes. The LLC passes a network ID to the device driver so the driver can route the receive data to the proper LLC. The device driver gets the network ID from the receive data and then determines (from the driver's network ID correlation table) the LLC to be notified, the device ring queue and LLC ring correlator to use.

A device or port may have more than one network ID associated with it. If multiple LLCs are attached to the same device, each LLC must have a unique network ID. The device driver discards any data which is received with an undefined network ID.

You can specify a network ID with a length of zero, but in this case only one entry can exist in the network ID correlation table and no network ID comparison is done. In this case, all data is routed to the one LLC.

The network ID correlation table is used by a device driver to associate a network ID to a device ring queue, path ID, and LLC process ID. Each time the device driver receives a unique network ID (from an LLC by way of a start device command) an entry is created in this table. An LLC may have multiple network IDs and may use the same network ID on multiple ports. The same network ID cannot be used by multiple LLCs on the same port.

Each port (IODN) has a network ID correlation table. The number of table entries and the network ID length are specified in the port's DDS. The following figure shows the network ID correlation table fields required by the block I/O subsystem. Additional user-defined fields may be added.

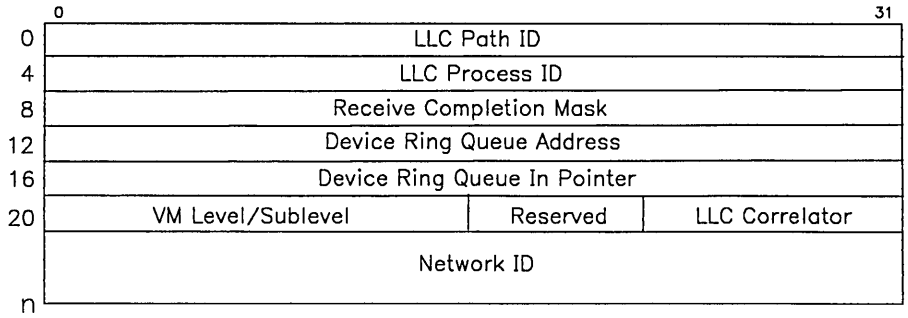


Figure 4-22. Network ID Correlation Table

The fields in the preceding figure are defined as follows:

- LLC Path ID — is the ID of the LLC-to-device driver path. A value of zero in this field indicates an unused entry.
- LLC Process ID — is the ID of the LLC that issued the start device command for this network ID
- Receive Completion Mask — is the ECB mask to use when notifying an LLC of receive data for a network ID. This field is used only by VRM LLCs.
- Device Ring Queue Address — is a pointer to the device ring queue
- Device Ring Queue In Pointer — is a copy of the In pointer contained in the device ring queue associated with this table entry. Because LLCs could disturb the In pointer in the device ring queue, the VRM device driver reads the In pointer value from this location, ensuring that the driver will never cause a page fault by accessing memory with a pointer made invalid by another process.
- VM Level/Sublevel — is the interrupt level and sublevel to be used when notifying an operating system LLC of receive data for this network ID or of a network failure. For operating system LLCs, bits 5-7 contain the interrupt level and bits 8-15 contain the interrupt sublevel. For VRM LLCs, this field is set to 0x0000.
- LLC Correlator — is the LLC ring correlator that the LLC passed with the start device command. This value is returned in all acknowledgment queue elements or PSBs to an LLC.
- Network ID — indicates the network ID passed on the start device command. The length of this field is indicated in the driver's DDS. If the network ID is zero, this field can be excluded from the structure.

SLIH Ring Queue

The SLIH ring queue contains a list of pinned receive data buffers to be used by the SLIH. Each port has a SLIH ring queue. The device manager puts the buffer addresses in the structure at **open** time, and the buffers are removed from the structure when the device driver adds them to the device ring queue. The DDS indicates the number of buffers that the SLIH ring queue can hold. The device driver must initialize the In, Out, and End pointers, the device manager process ID, and the ECB mask.

The device manager adds a buffer to the ring by writing the buffer address at the location indicated by the In pointer and then incrementing the In pointer to the next entry in the ring queue. The device driver removes a buffer from the ring by getting the buffer address at the location indicated by the Out pointer and then incrementing the Out pointer to the next entry in the ring queue.

The ring queue is in the empty state when the In pointer is equal to the Out pointer. The queue is full when the In pointer is one entry behind the Out pointer.

The following figure shows a SLIH ring queue.

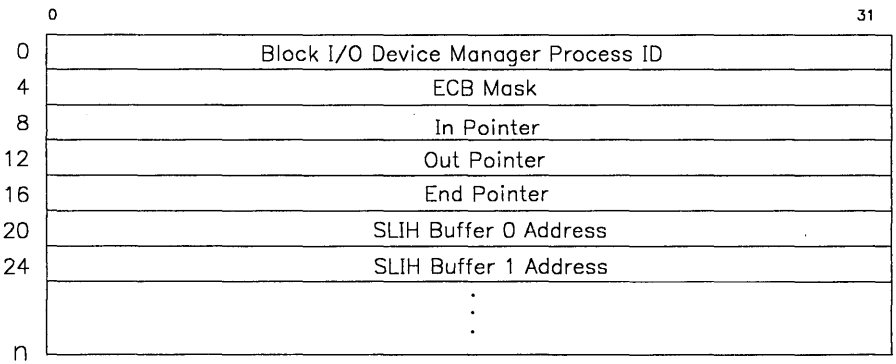


Figure 4-23. SLIH Ring Queue

The fields in the preceding structure are defined as follows:

- Device Manager Process ID — is the ID of the device manager
- ECB Mask — is the mask the driver uses with **_post** to request additional buffers for the SLIH ring queue from the device manager
- In Pointer — is the address in the SLIH ring queue at which the device manager should place the address of the next buffer
- Out Pointer — is the address in the SLIH ring queue that contains the address of the next buffer to be used by a device driver
- End Pointer — is the address in the SLIH ring queue of the last SLIH buffer address entry.
- SLIH Buffer Addresses — are addresses of pinned receive buffers that have been enqueued to the ring. The number of entries in this structure is one more than the number of SLIH ring queue buffers specified in the driver's DDS.

Block I/O Communication Area

A block I/O communication area must be built by the operating system for each port (IODN) at configuration time. The address of the area is passed to the device driver in the device's DDS.

The block I/O device manager obtains the address of the communication area for a device when it opens the device. The segment ID and segment offset of the buffer pool section is returned to the LLC by way of return parameters of the **open**. The LLC uses this returned information to obtain buffers from the buffer pool with the **_bfget** routine.

All areas of this structure, except for the area for the buffers, are pinned by the device manager when the device manager receives the first **open** request for the device driver IODN. The structure is unpinned when the manager receives the last **close** request for the IODN.

The device driver accesses the device ring queue array when it passes receive data to an LLC. The driver can access the structure only after the first **start device** request.

LLCs access the device queue ring array to obtain receive data and use the other areas to both obtain buffers and return buffers to the pool.

The structure of a block I/O communication area is shown in the following figure.

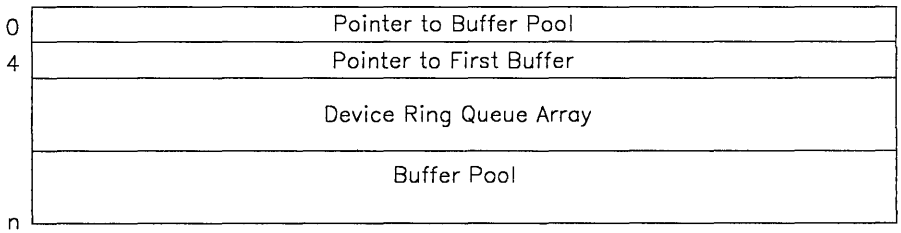


Figure 4-24. Block I/O Communication Area

The fields in the preceding figure are contiguous and must start on fullword boundaries. The fields are defined as follows:

- Pointer to Buffer Pool — indicates the address of the beginning of the buffer pool area
- Pointer to First Buffer — indicates the address of the beginning of the first buffer in the buffer pool
- Device Ring Queue Array — is a group of device ring queues used to pass data from a device driver to an LLC
- Buffer Pool — consists of a control area that indicates which buffers in the buffer area are used or unused and the buffer area itself.

The following sections provide more details on the fields of the block I/O communications area.

Device Ring Queue Array

The device ring queue array consists of a group of queues used to pass received data buffers from the device driver to LLC processes. Each network ID has a device ring queue. Each ring queue contains the addresses of receive data buffers that have been enqueued by the driver for the LLC. The device manager pins these buffers, the device driver unpins the buffers, and the LLC returns the buffers to the pool with the **_bffree** routine.

A device ring queue array is shown in the following figure.

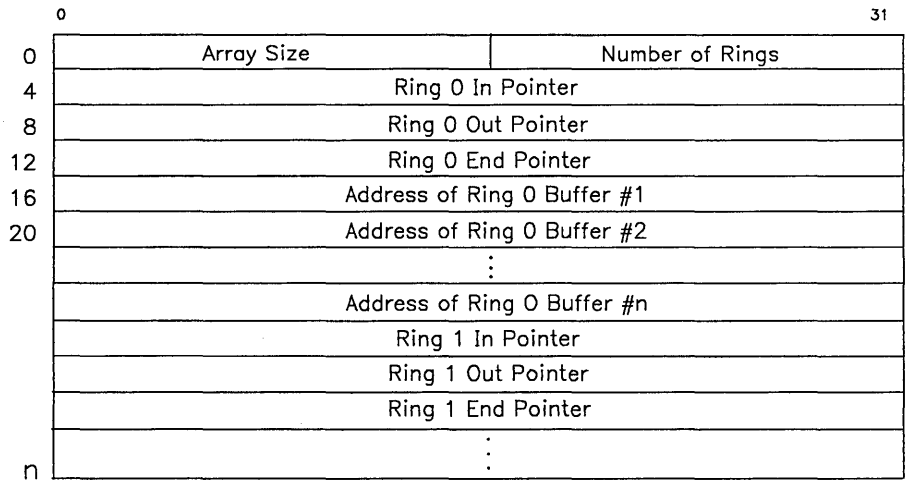


Figure 4-25. Device Ring Queue Array

The fields in the preceding figure are defined as follows:

- Array Size — is the length of the array in bytes
- Number of Rings — is the number of device ring queues in the array (this is the same as the number of concurrent network IDs the device driver supports)
- Ring 0 In Pointer — is the address in the first device ring queue at which the device driver should place the address of the next buffer
- Ring 0 Out Pointer — is the address in the first device ring queue that contains the address of the next buffer to be processed by an LLC and removed from the ring
- Ring 0 End Pointer — is the address in the first device ring queue of the last ring buffer address entry.
- Address of Ring 0 Buffers — is a series of receive buffer addresses that have been enqueued to the ring. The number of addresses is one more than the number of device ring queue buffers specified in the DDS
- Remaining Ring Queue — each additional ring queue has the same format as ring 0.

A driver adds a buffer to a ring by writing the buffer address at the location indicated by the In pointer and then incrementing the In pointer to the next entry in the ring queue. The LLC removes

a buffer by getting the buffer address at the location indicated by the Out pointer and then incrementing the Out pointer to the next entry in the ring queue. Each time a ring queue goes from an empty to a non-empty state, the device driver enqueues an unsolicited interrupt to an operating system LLC or does a **_post** to a VRM LLC.

The ring queue is at an empty state when the In pointer is equal to the Out pointer. The queue is full when the In pointer is one entry behind the Out pointer. If the device driver attempts to add a buffer to the ring and in doing so the In pointer equals the Out pointer, an overflow occurs. In this condition the entry is not saved. An unsolicited interrupt is generated when an overflow condition occurs.

Buffer Pool

The buffer pool is a control area that indicates which buffers in the buffer area are used and which are unused and the buffer area.

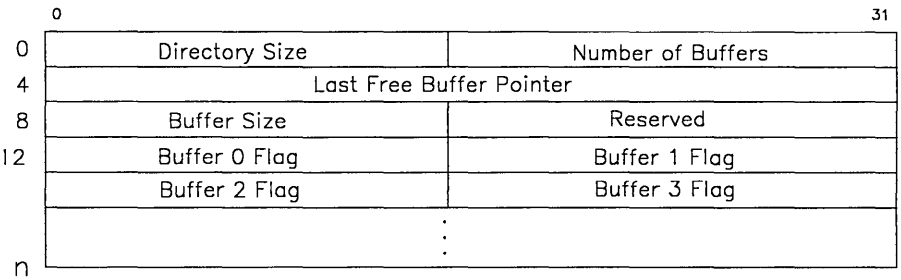


Figure 4-26. Buffer Pool Control Area

The fields in the preceding figure are defined as follows:

- Directory Size — is the length of the buffer pool in bytes
- Number of Buffers — indicates the number of buffers in the buffer pool
- Last Free Buffer Pointer — is the address of the buffer indicator that corresponds to the last buffer freed or the last buffer that was returned to the pool.
- Buffer Size — indicates the length in bytes of each buffer. All buffers in the buffer area have this length.
- Buffer Flags — are two-byte fields that indicate if a buffer is available or in use. Each buffer in the buffer pool has a corresponding flag. A value of 0xFF00 means the buffer is in use; a value of 0x0000 means the buffer is available.

Data Buffer Structure

The buffer area consists of data buffers. The first buffer begins at the start of the buffer area and each subsequent buffer begins where the last one ends. All buffers must be the same length, must begin on a fullword boundary, and must be a multiple of words in length.

The data buffer structure is shown in the figure on the following page.

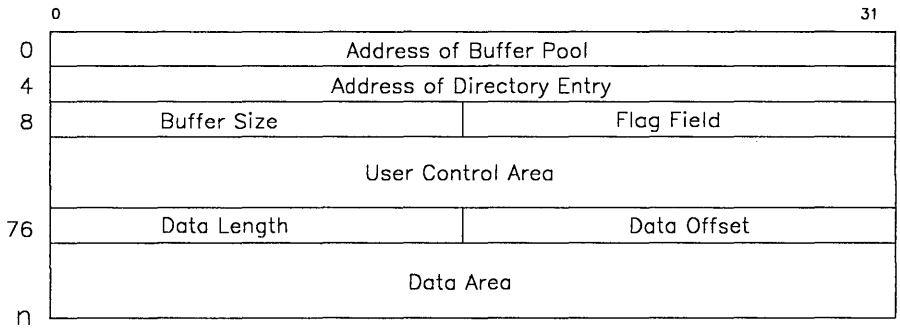


Figure 4-27. Data Buffer Structure

The fields in the preceding figure are defined as follows:

- Address of Buffer Pool — points to the beginning of the buffer pool that contains the buffer.
- Address of Directory Entry — is the address of the entry in the buffer pool control area for this buffer
- Buffer Size — is the size of the buffer in bytes. This value must be a multiple of 4 so buffers will hold fullwords of data.
- Flag Field — is a user-defined field with the exception of the low-order bit. The low-order bit is defined as the receive buffer overflow and is set to indicate that the buffer is not large enough to hold the receive data.
- User Control Area — a 64-byte user-defined field.
- Data Length — is the length of the data in bytes
- Data Offset — is the offset, in bytes, of the start of the data from the start of the data area. This value is set by the LLC for send data and by the device driver for receive data
- Data Area — is the area where the device driver places the receive data. In the case of a write, the device driver gets the data to be written from this area.

Operation Results

The operation results returned to acknowledge a command request or as an unsolicited interrupt are of the format shown in Figure 4-28 on page 4-37 for block I/O subsystem operations.

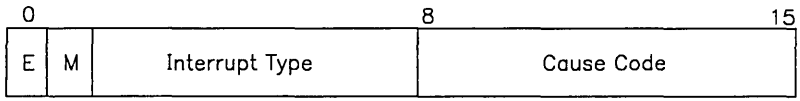


Figure 4-28. PSB Operation Results Field for Block I/O Requests

The fields in the preceding figure are defined as follows:

- E — indicates whether an error occurred.

  When this bit equals zero, the request was successful. When this bit equals one, an error occurred.
- M — indicates whether device driver errors occurred for **open** or **IPL adapter** requests. This bit is set by the block I/O device manager, not by the VRM device driver.
- Interrupt type — a 6-bit field that defines the error.
- Cause code — specifies the cause for the various interrupt types.

Device Driver-to-LLC Results

The following interrupt values are defined. Fields marked 'xx' can be defined by cooperating drivers and LLCs.

- 0x0000 = successful completion
- 0x8100 = receive data available
- 0x82xx = timeout error
- 0x83xx = device initialization failure
- 0x8400 = SLIH ring queue underflow
- 0x8500 = device ring queue overflow
- 0x8601 = network ID table full
- 0x8602 = duplicate network ID
- 0x8603 through 0x86FF = device-dependent network ID errors
- 0x8700 = buffer pool depleted
- 0x88xx = command parameter error
- 0x89xx through 0xBFxx = device-dependent error codes.

Block I/O Device Manager

This section defines the block I/O device manager. The block I/O device manager is a component of the block I/O subsystem. The primary components of this subsystem are ***logical link control*** (LLC) processes, a block I/O device manager, and the device drivers that control block I/O adapters. Examples of block I/O subsystem components are the Interface Program for TCP/IP (LLC), the Baseband Device Driver (VRM device driver), and Baseband Adapter (block I/O device adapter). See "Block I/O Device Driver Considerations" on page 4-5 for more information on this subsystem.

The DDS of the block I/O device manager is defined at IPL time by a loadable POST. Pertinent fields from the manager's DDS include:

- IODN = 0x300.
- IOCN = 0x310.
- Define options = 0x0001 (add).
- Device type = 0x0002 (shareable device).
- Device name = ignored by the VRM.
- Offset to hardware characteristics = 0 (device manager).
- Offset to device characteristics = 0 (device manager).
- Offset to error log = 28 bytes. This field is described in more detail in "Block I/O Device Manager Error Logging" on page 4-44.

The primary functions of the device manager are:

- To pin the receive buffers used by the block I/O device drivers
- To set up the communication paths between the LLCs and the device drivers.

The components of the block I/O subsystem are shown in the following figure.

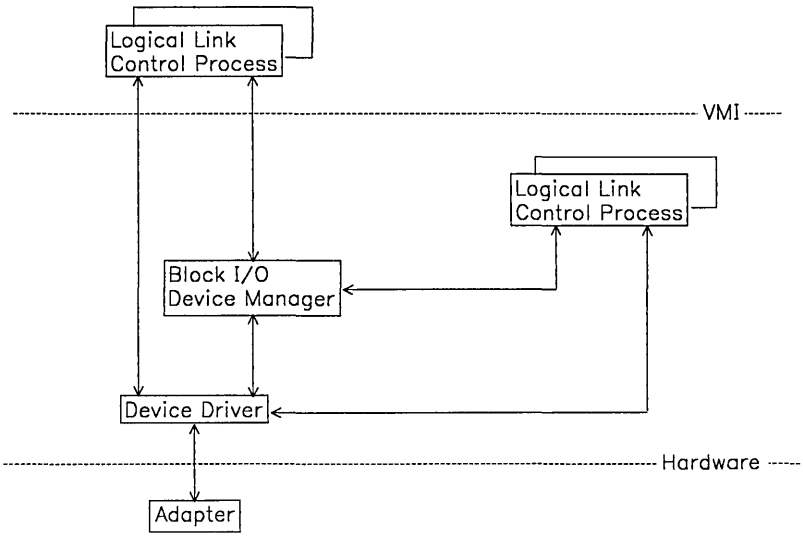


Figure 4-29. Block I/O Subsystem Components

The device manager can support a maximum of 22 IODNs (ports or devices) at a time.

To set up the paths necessary for data transfer between the subsystem components, the LLC must first attach to the device manager. Depending on the location of the LLC (in the operating system or VRM space), this is done with either an **Attach Device SVC** or the **_attchq** routine.

The LLC then issues an **open** command to the device manager. The LLC specifies the IODN of the device or port it wants to use in a Send Command queue element to the device manager. (For operating system LLCs, these queue elements are created by VRM SVC handlers and sent to the device manager). The IODN of the port or device opened is not returned to the LLC, but requests are acknowledged in the same order they are received by the device manager. LLCs that issue multiple **open** commands can therefore determine which commands completed by keeping track of the order in which commands were issued.

When the device manager receives the first **open** command from an LLC for a device driver's IODN, it attaches itself to the device driver with **_attchq**. The device manager then does the **open** to the device driver. When the device driver acknowledges the **open**, it may indicate in the operation results field of the acknowledge queue element that the adapter requires a separate IPL of adapter microcode. (See "Operation Results" on page 4-42 for more details.) If so, the device manager immediately issues the **IPL adapter** command.

When the **open** command is complete and the adapter is IPLed (if required), the device manager issues an **_attchq** to attach the LLC to the device driver. The LLC is now ready to send commands (such as **start device**, **halt device**, **write**, and so on) to the device driver.

The following sections provide more details on the specific programming interfaces.

LLC Process-to-Block I/O Device Manager Interface

The LLC process attaches to and detaches from the block I/O device manager to begin the process of accessing a specified device. As shown in Figure 4-29, LLC processes can reside in the operating system or in the VRM. The location of the LLC determines how the device manager interface is defined.

Interface to Operating System LLCs

For operating system LLCs, the interface to the device manager consists of the following SVCs:

- **Attach Device SVC** - to attach the virtual machine to the device manager
- **Send Command SVC**

  Input parameters and return codes for this SVC are defined in *VRM Programming Reference.* Pertinent values for the block I/O device manager include:
  - GPR2 = Bits 0-15 - device manager IODN (0x300); bits 16-31 - operation options

    The command extension bit of the operation options field must be zero. The synchronous operation bit should be set to zero. The following device options are defined:
    Open = 2
    Close = 3
  - GPR3 = interrupt level and sublevel to use when notifying the LLC (by way of a virtual interrupt) of a completed **Open** or **Close**.
  - GPR4 through GPR6 = reserved
  - GPR7 = path ID of the device manager-to-virtual machine path.

  Acknowledgments to an **open** or **close** request with the **Send Command SVC** are contained in the operation results field of the Send Command program status block. If the high-order bit of this field equals zero, the request was successful. If the high-order bit equals one, an error occurred. "Operation Results" on page 4-42 explains the errors.

  The Send Command PSB also contains the following values:
  - Byte offset 26 - two-byte segment ID of the device's buffer pool
  - Byte offset 28 - four-byte segment offset of the device's buffer pool
  - Byte offset 32 - four-byte ID of the virtual machine-to-device path.
- **Detach Device SVC** - to detach the virtual machine from the device manager. The detach should not be performed until all LLC processes owned by the virtual machine are finished using the device manager.

Interface for VRM LLCs

For VRM LLCs, the interface to the device manager consists of the following VRM runtime routines:

- **_attchq** to attach the LLC to the device manager
- **_enque**

  Input parameters and return codes for this routine are described in *VRM Programming Reference*. To perform an **open** or **close** request, the Send Command queue element is used. Figure 4-30 on page 4-41 shows the queue element for block I/O requests.

0	Reserved		
4	Path ID		
8	Type	Priority	Operation Options
12	IODN		Reserved
16	VM Interrupt Level/Sublevel		Device IODN
20	Maximum # Interrupts		
24	Queue ID		
28	Reserved		

Figure 4-30. LLC-to-Manager Send Command Queue Element

The fields in the preceding figure are defined as follows:

- Path ID - LLC-to-device manager path
- Priority - set to zero
- Operation Options - command extension bit must be set to zero. Low-order byte indicates the requested operation.
  Open = 2
  Close = 3
- IODN - device manager IODN (0x300)
- VM Interrupt Level/Sublevel - set to 0x0000 for VRM LLCs
- Device IODN - IODN of the device to be opened or closed
- Maximum # Interrupts - for the LLC-to-VRM device driver path. Note that this value affects the amount of memory pinned for the device. For a **close** request, this queue element field is reserved.
- Queue ID - is the ID of the LLC queue that will receive acknowledgments to device driver commands.

Acknowledgment of an **open** or **close** request by a VRM LLC is contained in the acknowledgment queue element shown in Figure 4-31 on page 4-42.

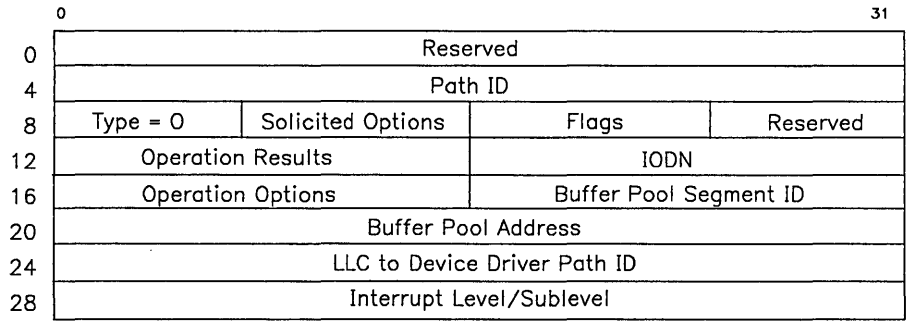


Figure 4-31. LLC-to-Manager Acknowledgment Queue Element

Pertinent queue element values for the device manager include:

- Path ID - LLC-to-device manager path
- Solicited Options - contains the high-order byte of the Operation Options field
- Flags - set to 0x14 to indicate a solicited acknowledgment to a Send Command request
- Operation Results - this field is defined in "Operation Results"
- IODN - block I/O device manager IODN (0x300)
- Operation Options - contains the Operation Options field of the input Send Command queue element
- Buffer Pool Segment ID - indicates the segment ID of the device's allocated buffer pool. This field is not used for acknowledgment to a **close** request.
- Buffer Pool Address - indicates the offset into the segment for the device's buffer pool. This field is not used for acknowledgment to a **close** request.
- LLC to Device Driver Path - contains the path ID of the LLC-to-device driver path. This field is not used for acknowledgment to a **close** request.
- Interrupt Level/Sublevel - set to zero for VRM LLCs.

- **_detchq** to detach the LLC from the device manager.

Operation Results

The operation results returned to acknowledge a command request or as an unsolicited interrupt are of the format shown in Figure 4-32 for block I/O subsystem operations.

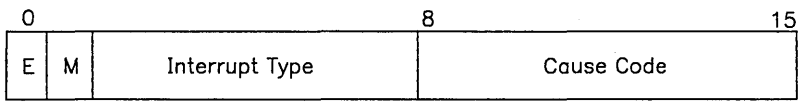


Figure 4-32. PSB Operation Results Field for Block I/O Requests

The fields in the preceding figure are defined as follows:

- E - indicates whether an error occurred.

  When this bit equals zero, the request was successful. When this bit equals one, an error occurred.
- M - indicates whether device driver errors occurred for **open** or **IPL adapter** requests. The M bit is set by the block I/O device manager; the remaining bits (Interrupt Type and Cause Code) indicate the error returned by the device driver.
- Interrupt type - a 6-bit field that defines the error.
- Cause code - specifies the cause for the various interrupt types.

Device Manager-to-LLC Results

The following interrupt values are defined:

- 0x0000 = successful completion
- 0x8800 = invalid queue element type
- 0x8801 = invalid command
- 0x8802 = invalid device IODN
- 0x8900 = tried to open more than 22 devices
- 0x8901 = tried to open a device already opened the maximum number of times allowed
- 0x8902 = another virtual machine has opened the device
- 0x8903 = device already opened by the VRM LLC
- 0x8904 = another component attached to the device before the device manager
- 0x8905 = tried to close a device that is not opened
- 0x8B00 = attach to device driver failed, probably because the assigned DMA channel or interrupt level conflicts with an existing device
- 0x8B01 = enqueue of command to device driver failed
- 0x8B02 = invalid DDS
- 0x8B03 = invalid microcode IOCN in DDS
- 0x8B04 = "IPLCODE" or "_IPLCODE" are unresolved symbols in the microcode
- 0x8B05 = unable to pin device ring queues and buffer pool control area in the communication area
- 0x8B06 = unable to fill device's SLIH ring queue
- 0x8B07 = unable to attach LLC to device driver
- 0x8C00 = unable to unpin SLIH ring queue buffers, device ring queues, or buffer pool control area.
- 0xCxxx = device driver error on open or IPL adapter command. See the description of device driver error 0x8xxx for details.
- 0xDxxx = device driver error on open or IPL adapter command. See the description of device driver error 0x9xxx for details.
- 0xExxx = device driver error on open or IPL adapter command. See the description of device driver error 0xAxxx for details.
- 0xFxxx = device driver error on open or IPL adapter command. See the description of device driver error 0xBxxx for details.

VRM Device Driver-to-Device Manager Results

The only valid interrupt type returned by the device driver to the device manager is 000000 (successful). Any other type causes the device manager to set the 'M' bit and return the device driver's operation results to the LLC.

If the device manager receives an unsolicited response from a device driver, both the interrupt type and cause code are ignored.

Cause codes defined for interrupt type 000000 (successful) are:

- 0x00

  This cause code is returned for all **IPL adapter** commands, as well as for an **open** command requiring no adapter IPL.
- 0x01

  This cause code is returned for **open** commands that also require an adapter IPL.

Block I/O Device Manager Error Logging

The format of the error log section of the block I/O device manager DDS is shown in Figure 4-33.

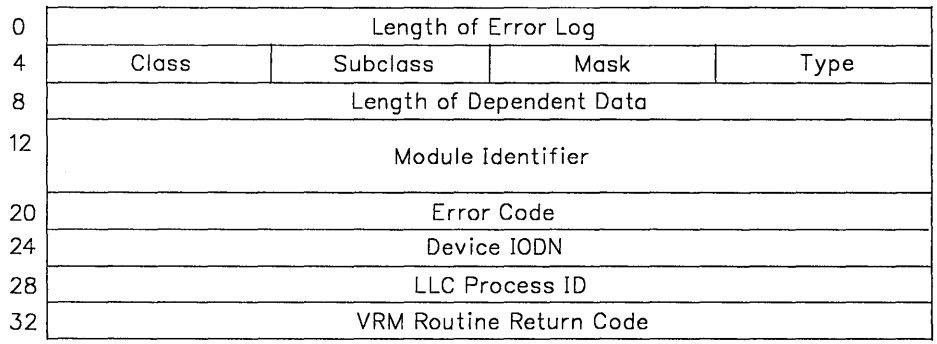


Figure 4-33. Block I/O Device Manager Error Log Format

The fields in the preceding figure are defined as follows:

- Length of Error Log = 9 words
- Class = 0x02 (software error)

- Subclass = 0x07 (VRM component)
- Mask = 0x06 (block I/O device manager error)
- Type = 0x80 (permanent error)
- Length of Dependent Data = 0x07 words
- Module Identifier

  This 8-character field identifies the block I/O device manager and is set to 'BIO_DM '.
- Error Code

  Bits 0-15 of this word are reserved; bits 16-31 specify the actual error that occurred. The following values are defined:
  0x0001 = Unexpected queue element received from device driver.
  0x0002 = Unable to fill device's SLIH ring queue with pinned receive buffers when requested by device driver.
  0x8B00 = Unable to attach to a device when the device was opened.
  0x8B01 = Unable to enqueue command to device driver.
  0x8B02 = Invalid DDS device characteristics section for device being opened.
  0x8B03 = Device driver that requires microcode load has an invalid IOCN for the microcode in its DDS.
  0x8B04 = Unable to bind microcode of a device driver that requires microcode load because "IPLCODE" or "_IPLCODE" symbol not resolved.
  0x8B05 = Device's ring queue array could not be pinned.
  0x8B06 = Unable to fill device's SLIH ring queue when the device was opened.
  0x8B07 = Unable to attach a device to the LLC process that tried to open the device.
- Device IODN

  Bits 0-15 of this word are reserved; bits 16-31 contain the IODN of the device associated with the error. For example, for error code 0x0001, the IODN is of the device that sent the unexpected queue element to the manager. For 0x0002, the IODN is of the device requesting the buffers. For the rest of the errors, the IODN is of the device being opened or closed.
- LLC Process ID

  For all errors but 0x0001 and 0x0002, this field contains the ID of the LLC process attempting to open or close a device. For 0x0001, this word contains the operation results (bits 0-15) and operation options (bits 16-31) of the unexpected queue element. This field is not valid for error code 0x0002.
- VRM Routine Return Code

  Some of the errors logged by the block I/O device manager are due to errors from the VRM runtime services called by the manager. Depending on the error code, the manager may also log the return code from the unsuccessful VRM service. For the defined error codes, the following values are valid:

  0x0001 – not applicable.

  0x0002 – This field contains one if there were not enough buffers in the device's buffer pool to fill the SLIH ring. Otherwise, the ring could not be filled because a buffer could not be pinned. In this case, this field contains the return code from the **_pinpgs** call.

0x8B00 – contains the return code from the **_attchq** call.

0x8B01 – contains the return code from the **_enque** call.

0x8B02 – not applicable.

0x8B03 – contains the return code from the **_querym** call.

0x8B04 – contains the return code from the **_bind** call.

0x8B05 – contains the return code from the **_pinpgs** call.

0x8B06 – This field contains one if there were not enough buffers in the device's buffer pool to fill the SLIH ring. Otherwise, the ring could not be filled because a buffer could not be pinned. In this case, this field contains the return code from the **_pinpgs** call.

0x8B07 – contains the return code from the **_attchq** call.

Baseband Device Driver

The baseband device driver is a component of the block I/O subsystem. This device driver can handle two adapters at a time, with each adapter supporting one port (one IODN). Only one virtual machine can use an adapter at a time.

For additional information on block I/O device drivers, see "Block I/O Device Driver Considerations" on page 4-5.

Define Device Header

The define device structure (DDS) for the baseband device driver is created at configuration time and passed to the driver by the **Define Device SVC**. Each active adapter must have its own DDS.

Figure 2-2 on page 2-5 shows the format of a DDS.

Pertinent values for the baseband device driver are defined as follows:

IODN: This field contains a number that identifies the device for I/O activity.

IOCN: This field contains a number that links the device to the code that supports it.

Define options: This field contains an indication of the device definition options.

Device type: 0x0003

Define name: This field is ignored by the VRM, but it can be used to create convention names that identify the device driver.

Offset to hardware characteristics: 28 bytes

Offset to device characteristics: 80 bytes

Offset to error log: 0 bytes.

Hardware Characteristics

The format of this field is shown in Figure 2-3 on page 2-6. Values for the baseband device driver are defined as follows:

Length: 13 words
Internal device type:
0x11xx4000 - indicates the first adapter
0x11xx4001 - indicates the second adapter.
(where xx is set to the slot occupied by the adapter).

I/O port address (base):
0x0000FFFF

I/O port addresses (number):
Set to zero (not used).

Bus memory start address (RAM):
This contains the contents of the Bus Memory Start Address (ROM) plus 0x2000.

Bus memory end address (RAM):
This contains the contents of the Bus Memory Start Address (ROM) plus 0x7FFF.

DMA type:
Set to zero (not used).

Interrupt type:
0x800002xx

(where xx is an interrupt level, 03 through 05, 07 and 09, as set by jumpers on the adapter card).

Bus memory start address (ROM):
This address is determined by jumpers on the adapter card. The following addresses are valid:

0x80000	0x88000	0x90000	0x98000
0xA0000	0xA8000	0xB0000	0xB8000
0xC0000	0xC8000	0xD0000	0xD8000
0xE0000	0xE8000	0xF0000	0xF8000

Bus memory end address (ROM):
This is the contents of the Bus Memory Start Address (ROM) + 0x1FFF.

Device Characteristics

Figure 4-21 on page 4-29 shows the device characteristics fields for a block I/O device driver.

The values for a baseband device are defined as follows:

Length of Device Characteristics Section:
8 words

Communications Area Segment ID:
The value is placed in the DDS by the configuration process after construction of the communications area.

Communications Area Address:
The value is placed in the DDS by the configuration process after construction of the communications area.

Buffer Size:
This is a user-specified value placed in the DDS by the configuration process that gives the length of the individual buffers in the buffer pool. For information on the buffer size, see "Data Buffer Structure" on page 4-50.

Number of Buffers:
This is a user-specified value placed in the DDS by the configuration process that gives the number of buffers in the buffer pool.

Device Ring Buffer:
This is a user-specified value placed in the DDS by the configuration process that gives the number of buffers to be put on each device ring queue.

SLIH Ring Buffers:
This is a user-specified value placed in the DDS by the configuration process that gives the number of receive buffers to be put on the driver's SLIH ring queue.

Maximum # LLCs:
This is a user-specified value placed in the DDS by the configuration process that gives the maximum number of LLCs that may attach to the device driver at one time.

NetID Length:
The length, in bytes, of the device driver network ID. This value is fixed at 2 bytes.

NetID Pointer:
This is a user-specified value placed in the DDS by the configuration process that gives the offset, in bytes, to the received packet of the network ID. This value is fixed at 12 bytes.

Maximum # NetIDs:
This is a user-specified value placed in the DDS by the configuration process that gives the maximum number of Network IDs that the device driver can support.

Data Offset:
This value indicates an offset (in bytes) into the data area of the receive buffer at which the device driver is to place received data. The baseband device driver requires a value in this field.

SLIH Threshold:
This is a user-specified value placed in the DDS by the configuration process that indicates the number of SLIH ring queue buffers that the device driver can use before requesting additional buffers from the device manager.

SLIH Count:
The number of SLIH ring queue buffers that the device driver used. It is initially set to 0 and is incremented each time a buffer is moved from the SLIH ring queue to a device ring queue. When the value reaches the SLIH threshold, the device driver resets the field to 0 and issues a **_post** to the device manager.

Microcode IOCN:
Set to zero (not used).

SLIH Ring Address:
This is the address of the driver's SLIH ring queue. It is filled in by the device driver on an **open** command. The segment ID of the SLIH ring queue is the same as the DDS.

Block I/O Device Ring Queue

Each network ID has one device ring queue. This queue is described in "Device Ring Queue Array" on page 4-34.

Data Buffer Structure

All buffers must begin on a fullword boundary, must be a multiple of words in length, and must be large enough to hold the maximum size packet, the header, and a data offset value as specified in the device characteristics section of the baseband DDS. For more information, see **_bfget** in *VRM Programming Reference*. The data buffer structure is shown in Figure 4-27 on page 4-36. Specific values for the baseband device driver include:

Flag Field: This is a user-defined field with the exception of the low-order bit. The low-order bit is defined as the receive buffer overflow and is set to indicate that the buffer is not large enough to hold the receive packet. If the receive buffer overflow bit is set, the reminder of the packet is discarded.

User Control Area:
This area is 64 bytes in length and is not used by the baseband device driver.

Data offset: This value is set to the value contained in the Data Offset field of the baseband device driver DDS device characteristics section.

The data area must be large enough to contain a packet of data of the maximum size plus the value of the data offset field specified in the device characteristics section of the baseband DDS.

LLC Process-To-Baseband Device Driver Interface

The baseband device driver uses the LLC-to-device driver interface (the **Send Command SVC** and **Start I/O SVC** for OS LLCs and the **_enque** routine with Start I/O or Send Command queue elements for VRM LLCs) described in "LLC Process-to-Device Driver Interface" on page 4-12. Some additional baseband-specific information, including a baseband-specific device option, is defined in the following section.

Start Device

For OS LLCs that interface to the baseband device driver, a command extension of a least 10 bytes is always used. Bit 19 of GPR2 is therefore set to one. The command extension area is defined as follows:

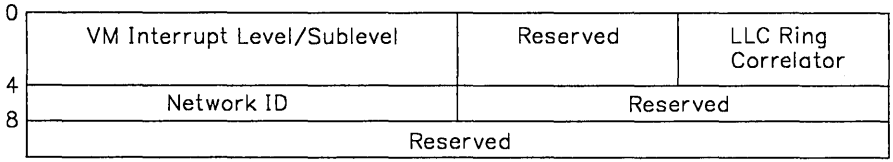


Figure 4-34. Command Extension for the Start Device Command

VM Interrupt Level/Sublevel:
: This is the level and sublevel on which virtual interrupts are returned to the LLC process.

LLC Ring Correlator:
: Byte value assigned by the LLC Process. All virtual interrupts returned for the following network ID contain the LLC ring correlator as a return parameter.

Network ID:
: A two-byte value that defines how the LLC identifies itself on the network.

The returned PSB information is described in "Start Device" on page 4-13. The format of the returned command extension is shown in the following figure.

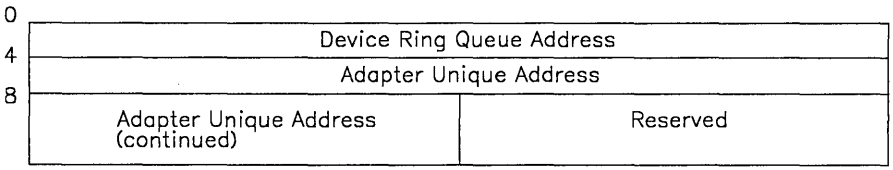


Figure 4-35. Returned Command Extension for the Start Device Command

Device Ring Queue address:
: This is the segment offset of the Device Ring Queue assigned to the Network ID passed on the **Start Device** command. The segment ID is the same as for the buffer pool and is not returned.

Adapter unique address:
: This 6-byte field contains the address of the adapter. The adapter reads this field and returns it to the LLC for use in the addressing on packets.

For VRM LLCs that interface to the baseband device driver, a command extension is always used. The input queue element, command extension, and acknowledgment queue element are defined in "Start Device" on page 4-13.

Halt Device

The **Halt Device** command ends a session with the device driver. This interface is defined in "Halt Device" on page 4-16.

Write

The **write** command is issued by an LLC that wishes to send data. The baseband device driver supports the **write long** and **start I/O write** forms of this command. See "Write Long" on page 4-18 and "Write (Start I/O)" on page 4-21 for interface definitions.

Query Statistics

The **query statistics** command is used by the LLC to read the counter values accumulated by the device driver. The counters are initialized to 0 by the **open** command and are cleared by setting bit 21 of the operation options field.

OS LLCs use the **Send Command SVC** to issue this command. Input parameters are:

- GPR2 = IODN for the desired adapter (bits 0-15), operation options (bits 16-31)
  - Bit 16 = Interrupt on completion (optional)
  - Bit 17 = Interrupt on error (optional)
  - Bit 18 = Synchronous operation (optional)
  - Bit 19 = Command extension (always 1)
  - Bit 20 = Reserved
  - Bit 21 = Clear counters after **query statistics**
  - Bits 22-23 = Reserved
  - Bits 24-31 = 9 for query statistics.
- GPR3 = Reserved
- GPR4 = Reserved
- GPR5 = Command extension address
- GPR6 = Command extension length (minimum of 64 bytes)
- GPR7 = ID of the LLC-to-device driver path.

The command extension passes additional information as shown in the following figure.

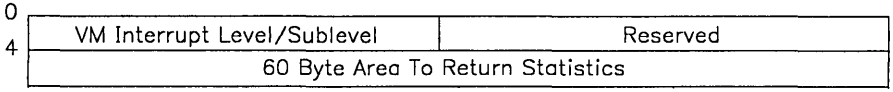


Figure 4-36. Command Extension for Query Statistics Command

Returned information for the SVC is contained in GPR2. Additional information is contained in the PSB and returned command extension.

Program status block completion information includes:

Status flag: 0x14 to indicate a solicited interrupt

Overrun count: Set to 0.

Operation results: Bits 0-15, see "Operation Results" on page 4-57; bits 16-31 contain the IODN for the baseband adapter.

Data word 1: Operation options (bits 0-15); segment ID of the command extension (bits 16-31).

Data word 2: Command extension address.

Data word 3: Reserved (set equal to 0).

The returned command extension area is defined as follows:

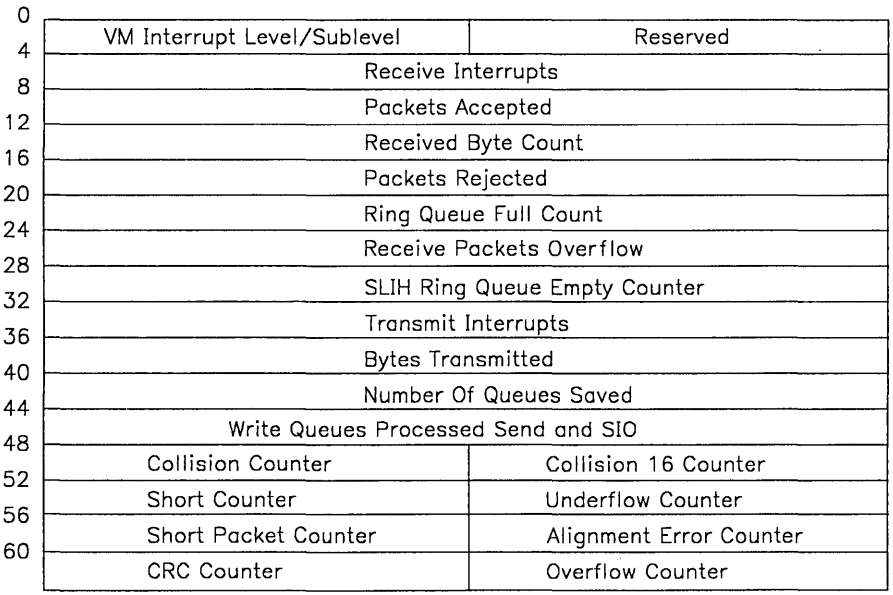


Figure 4-37. Returned Command Extension for the Query Statistic Command

VM Interrupt Level/Sublevel:
The interrupt level and sublevel of the LLC that initiated the **query statistics** command.

Receive Interrupts:
Total number of receive interrupts processed.

Packets Received:
Total number of packets accepted.

Note: Multiple packets may be received for each receive interrupt, or no packets may be received in the case of a receive error. Thus, the total number of packets received may not be the same as the total number of receive interrupts.

Receive Byte Count:
Total number of bytes received and placed in the receive buffer.

Packets Rejected:
Total packets rejected, no NetID match.

Ring Queue Full Count:
Total number of times any device ring queue went full and a **_post** or unsolicited interrupt was sent to the LLC. A packet is lost each time this occurs.

Receive Packet Overflows:
Total number of packets received that are too large for the receive buffer. Only the overflow data is lost.

SLIH Ring Empty Counter:
Total number of times the SLIH ring queue went empty. This indicates that a reconfiguration may be necessary to allocate more buffers, increase the SLIH ring queue size, or to free up received buffers sooner.

Transmit Interrupts:
Total number of Transmit Complete interrupts.

Bytes Transmitted:
Total number of bytes transmitted.

Number Of Queues Saved:
Total number of times it was necessary to save a queue element. Both transmit buffer were full. This is not an error condition.

Write Queues Processed Send and SIO
Total number of write queues processed.

Collision Counter:
Total number of packets that had at least one collision on transmit.

Collision 16 Counter:
Total number of packets that had 16 collisions on transmit. The data packet was not transmitted.

Short Counter:
Total number of electrical shorts detected between a signal wire and ground.

Underflow Counter:
Total number of underflow detects. Transmission is aborted due to a hardware failure.

Short Packet Counter:
Total packets received with less than 64-bytes (including the 4-byte CRC). This is a receive error counter.

Alignment Error Counter:
Total number of alignment errors detected. The packet size is not an integer multiple of 8 bits. This is an adapter line receive error counter.

CRC Counter:
Total number of CRC error detects. This is a receive error counter.

Overflow Counter:
Total number of overflows detected. This is a receive error counter that indicates that the receive buffer is full and that the data packet from the network was lost.

Note: Multiple receive errors of the same type may accumulate before the interrupt is processed; only one error would be indicated by the adapter. Thus, there may be other receive errors that are not indicated in the receive error counters.

VRM LLCs use the Send Command queue element to issue the **query statistics** command. The input queue element is defined as follows:

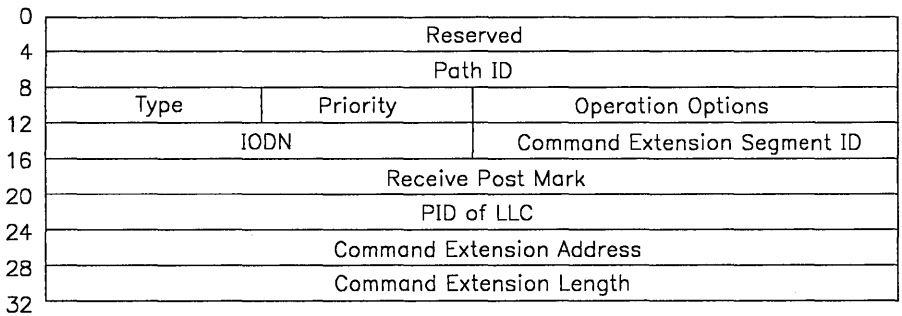


Figure 4-38. Query Statistics Queue Element

- Path ID - ID of the LLC-to-device driver path.
- Type - set to 1 for Send Command queue element.
- Priority - set to 0.
- Operation options - The command extension bit of the operation options field must be 1. The device option field is set to 9.

- Bit 16 = Interrupt on completion (optional)
- Bit 17 = Interrupt on error (optional)
- Bit 18 = Synchronous operation (optional)
- Bit 19 = Command extension (always 1)
- Bit 20 = Reserved
- Bit 21 = Clear counters after **query statistics**
- Bits 22-23 = Reserved
- Bits 24-31 = 9 for query statistics.

- IODN - of the adapter.
- Command Extension Segment ID - segment ID of the command extension area.
- Receive Post Mask - the ECB bit used when the device driver **_posts** the LLC to indicate received data.
- PID of LLC - the process ID of the LLC.
- Command Extension Address - the address of the command extension area.
- Command Extension Length - the length of the command extension

The command extension passes additional parameters (see Figure 4-36 on page 4-53). For VRM LLCs, the VM Interrupt Level/Sublevel filed is set to 0.

Returned information for **_enqueue** is described in *VRM Programming Reference*. Additional returned information is contained in the returned command extension (see Figure 4-37 on page 4-53) and the acknowledgement queue element.

The acknowledgement queue element is defined as follows:

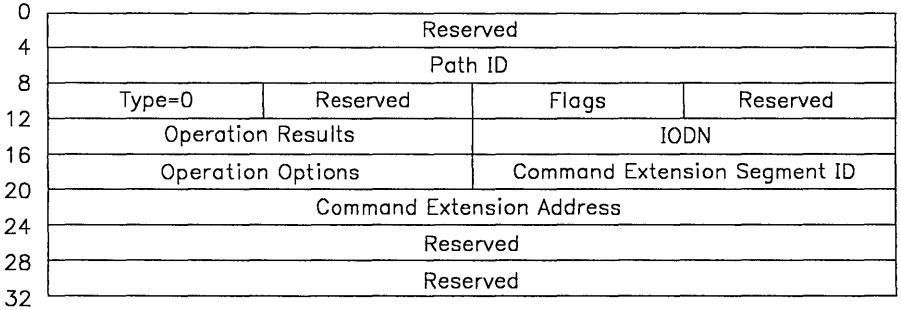


Figure 4-39. Acknowledgment Queue Element for the Query Statistics Command

Path ID: ID of the LLC-to-device driver path.

Type: Set to 0 for acknowledgement queue element.

Flags: 0x14 to indicate a solicited acknowledgement to a Send Command

Operation results:
See "Operation Results" on page 4-57.

IODN: Of the adapter.

Operation Options:
The operation options field from the Send Command queue element.

Command Extension Segment ID:
Segment ID of the command extension area.

Command Extension Address:
Address of the command extension area.

Receive Data

When the address of a packet matches the address of an adapter or of a multicast or broadcast address, the adapter receives that packet and places it in the adapter receive buffer.

Note: The IBM RT PC Baseband VRM device driver does not support promiscuous addressing.

When a packet is placed in the receive buffer, an interrupt is generated. The device driver checks the netID in the type field for a match in the network ID correlation table. If the device driver does not find a match, the packet is purged from the adapter receive buffer. If a match is found, the data moves to a receive buffer of the SLIH ring queue. The data is moved to an offset in the receive buffer as specified by the Data Offset value contained in the device characteristics section of the driver's DDS.

The buffer address is then placed in the device ring queue. If the device ring queue was empty, the LLC is informed that a buffer was received. For VRM LLCs, a **_post** routine is used. For OS LLCs, a virtual interrupt is used. If the ring queue was not empty, no action is taken as the LLC is expected to empty the ring queue once it is informed that buffers are available.

Operation Results

Operation results for a command or unsolicited interrupt are defined in "Operation Results" on page 4-42.

The acknowledgement by the device driver to an LLC for the **start device**, **halt device**, and **write** commands are always solicited. These operation results are defined as follows:

0x0000: Successful operation
0x8200: Timeout error on transmit
0x8601: NetID table full
0x8602: Duplicate NetID in table
0x8801: Invalid queue element type
0x8802: Invalid command

0x8803: Invalid parameter
0x8804: Port is not open.

If there is receive data for an LLC or an error occurs that is not related to the specified command, the device driver may send an unsolicited acknowledgement queue element or interrupt to the LLC. Operation results that the device driver returns for these conditions are as follows:

0x8100: Data available
0x8400: SLIH ring queue empty

This indicates that a reconfiguration may be necessary to allocate more buffers, increase the SLIH ring queue size, or to free received buffers sooner.

0x8500: Device ring queue full

This indicates that the LLC process is not keeping up with the receive data. A packet of data is lost for each occurrence of this result.

The device driver always sends solicited acknowledgements to the block I/O device manager for an **open** command. These operation results are defined as follows:

0x0000: Successful return code
0x8805: Port is already open.

Unsolicited queue elements are never sent to the device manager by the device driver.

IBM PC 3278/79 Emulation Adapter Distributed Function Terminal Device Driver

The distributed function terminal (DFT) device driver is a component of the block I/O subsystem. This device driver supports as many as four communications adapter cards at a time, with one port per adapter. Each port has its own define device structure (DDS) and supports up to eight sessions at a time.

If the attached control unit does not support **Extended Asynchronous Event Device Status**, the control unit port must be configured for one session only or the device driver must be configured for one session only.

The 3278/79 Emulation Advanced Adapter is the communications adapter supported by the DFT driver. The 3278/79 emulation adapter is available in two forms, a long card (part number 1602507) and a short card (part number 8665789). Note that the DFT device driver supports only the short card.

The DFT device driver supports terminals that use the BSC or non-SNA channel protocols. The driver does not support SNA.

For additional information on the block I/O subsystem, see "Block I/O Device Driver Considerations" on page 4-5 and "Block I/O Device Manager" on page 4-38.

Define Device Header

The define device structure (DDS) for this device driver is created at configuration time and passed to the driver by the **Define Device SVC**.

Figure 2-2 on page 2-5 shows the format of a DDS.

Pertinent values for the DFT device driver are defined as follows:

IODN: This field contains a number that identifies the device for I/O activity.

IOCN: This field contains a number that links the device to the code that supports it.

Define options:
This field contains an indication of the device definition options.

Device type:
0x0003

Define name:
This field is ignored by the VRM, but it can be used to create convention names that identify the device driver.

Offset to hardware characteristics:
28 bytes

Offset to device characteristics:
72 bytes

Offset to error log:
192 bytes.

Hardware Characteristics

The format of this field is shown in Figure 2-3 on page 2-6. Values for the DFT device driver are defined as follows:

Length: 11 words.

Internal device type:
0xD1xx3401

(where xx indicates the slot occupied by the adapter).

I/O port address (base):

0x000002D0 – first adapter
0x000006D0 – second adapter
0x00000AD0 – third adapter
0x00000ED0 – fourth adapter.

I/O port addresses (number):
0x0000000B.

Bus memory start/end addresses (RAM):
Bus memory addresses can be assigned to any 8K-byte sector within a 1 M-byte range. The default addresses are defined as follows:

- First adapter
  Start – 0x000CE000
  End – 0x000CFFFF
- Second adapter
  Start – 0x000D0000
  End – 0x000D1FFF
- Third adapter
  Start – 0x000D2000
  End – 0x000D3FFF
- Fourth adapter
  Start – 0x000D4000
  End – 0x000D5FFF

DMA type: Set to 0 (not used).

Interrupt type:
0x80000209

The rest of the fields in this section are reserved (set to zero).

Device Characteristics

Figure 4-21 on page 4-29 shows the device characteristics fields for a block I/O device driver.

The values for the DFT device driver are defined as follows:

Length of Device Characteristics Section:
30 words

Communications Area Segment ID:
The value is placed in the DDS by the configuration process after construction of the communications area.

Communications Area Address:
The value is placed in the DDS by the configuration process after construction of the communications area.

Buffer Size:
This is a user-specified value placed in the DDS by the configuration process that gives the length of the individual buffers in the buffer pool. This value must be a multiple of 4 so buffers will contain fullwords of data.

Number of Buffers:
This is a user-specified value placed in the DDS by the configuration process that gives the number of buffers in the buffer pool.

Device Ring Buffer:
This is a user-specified value placed in the DDS by the configuration process that gives the number of buffers to be put on each device ring queue.

SLIH Ring Buffers:
This is a user-specified value placed in the DDS by the configuration process that gives the number of receive buffers to be put on the driver's SLIH ring queue.

Maximum # LLCs:
This is a user-specified value placed in the DDS by the configuration process that gives the maximum number of LLCs that may attach to the device driver at one time. For the DFT device driver, this value is one.

NetID Length:
The length, in bytes, of the device driver network ID. The DFT driver uses a null NetID, so this value is fixed at zero.

NetID Pointer:
Because the DFT device driver uses a null NetID, this value is set to zero.

Maximum # NetIDs:
Indicates the maximum number of network IDs that the DFT device driver supports. This user-configurable value must be between 1 and 8. The value should be less than or equal to the number of sessions for which the control unit is configured.

SLIH Threshold:
This is a user-specified value placed in the DDS by the configuration process that indicates the number of SLIH ring queue buffers that the device driver can use before requesting additional buffers from the device manager.

SLIH Count:
The number of SLIH ring queue buffers that the device driver used. It is initially set to zero and is incremented each time a buffer is moved from the SLIH ring queue to a device ring queue. When the value reaches the SLIH threshold, the device driver resets the field to zero and issues a **_post** to the device manager.

Microcode IOCN:
Set to zero (not used).

SLIH Ring Address:
This is the address of the driver's SLIH ring queue. It is filled in by the device driver on an **open** command. The segment ID of the SLIH ring queue is the same as the DDS.

Figure 4-40 on page 4-63 shows additional device characteristics fields used by the DFT device driver. The fields in Figure 4-40 are defined as follows:

Machine Type:
This required field indicates the machine type of the machine that is running the 3278/79 emulation program. Possible values are the EBCDIC numerals 6150 or 6151, right-justified.

Customer ID:
This byte contains a required value that provides more information about the machine. The only valid values (all other values are reserved) are defined on the next page.

Bits 0-3 = 0xE for a customer-programmable machine

Bits 4-7 = 0x1 to indicate an IBM product.

Model Number:
This optional field can be used to further identify the machine. Possible values include all EBCDIC alphanumeric characters, right-justified and padded with 0x40 characters. If this value is not used or unknown, it is set to zero.

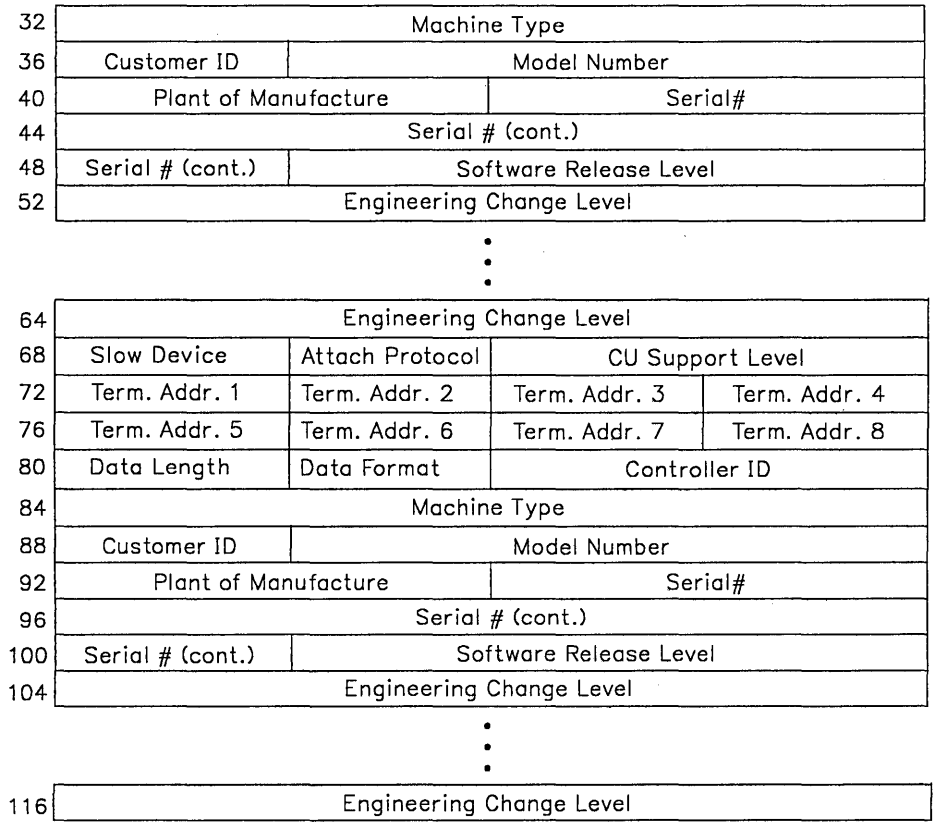


Figure 4-40. Additional DFT Device Characteristics Fields

Plant of Manufacture:
This optional field can be used to identify where the machine was produced. Possible values include all EBCDIC alphanumeric characters, right-justified. If this value is not used or unknown, it is set to zero.

Serial Number:
This optional field can be used to further identify the machine. Possible values include all EBCDIC alphanumeric characters, right-justified and padded with 0xF0 characters. If this value is not used or unknown, it is set to zero.

Software Release Level:
This optional field can be used to further identify the level of software used on the machine. Possible values include all EBCDIC alphanumeric characters, right-justified and padded with 0xF0 characters. If this value is not used or unknown, it is set to zero.

Engineering Change Level:
Bytes 52 through 67 can optionally be used to keep track of engineering change levels to the adapter card, control unit, and so on. Possible values include all EBCDIC alphanumeric characters, with user-defined justification and padding. If this field is not used or unknown, it is set to zero.

Slow Device:
Bit 0 of this byte indicates whether the device driver is allowed to respond to the control unit as if the terminal is a slow device. This support would be required for the emulator to run slow devices, such as certain ASCII terminals. Note that the control unit must provide the same level of support for slow devices to work properly. When bit 0 is set to 1, the device driver supports slow devices. When bit 0 is set to 0, the device driver waits for an immediate response from the emulator when it receives data. Bits 1-7 are reserved.

Attach Protocol:
Bits 0 and 1 of this byte contain information on the type of attachment and protocol in use for the control unit and host. Bits 2 through 7 of this byte are reserved (set equal to zero). When bit 0 = 0, the control unit is TP attached to the host; when bit 0 = 1 the control unit is locally attached. When bit 1 = 0, the protocol in use between the control unit and host is SNA; when bit 1 = 1, the protocol is non-SNA.

CU Support Level:
This two-byte field indicates the terminal control area options supported by the control unit and is valid after successful completion of an **open** command. Note that multiple bits of this field can be set simultaneously, indicating multiple options in effect. Bits 0 through 9 of this field are reserved (set equal to zero). When bits 10 through 16 of this field are set to 0, base terminal control area support is provided. The following options are in effect when the specified bit is set to 1:

Bit 10: AEEB extension for END busy.

Bit 11: Enhanced terminal control area buffer management.

Bit 12: Device-initiated UNBIND.

Bit 13: DFT network support

Bit 14: Read multiple (32 bytes)

Bit 15: Non-SNA slow device support/extended AEDV.

Term. Addr. 1 through 8:
Terminal addresses 1 through 8 indicate the session addresses configured at the control unit for this port. These fields are valid after successful completion of an **open** command. The five low-order bits of of each byte contains the address assigned by the host. A value of 0xFF in any byte indicates an undefined address.

The rest of the device characteristic fields (beginning with Data Length and ending with four additional words of Engineering Change Level information) are optionally passed by a control unit to the device after the first successful completion of a **Start device** command. These fields are defined as follows:

Data Length:
This field indicates the length of the data (in bytes) that the control unit has sent to the device. If the length is greater than 40 bytes, the device truncates the remaining bytes and places the first 40 bytes in this structure. If the length is zero, WCUS (20) may not have been received from the control unit yet. In this case, a session should be activated first or the transfer should be tried again later.

Data Format:
This field indicates whether the information presented was generated by the control unit or the device. Information generated by the control unit is identified by the value 0x00; all other values indicate information generated by the device.

Controller ID:
This field contains unique EBCDIC identification characters assigned to the various control unit types.

Machine Type:
This field contains right-justified numeric EBCDIC characters (such as 3174 or 3274) that identify the machine type on which the emulator is being run.

Customer ID:
This byte contains a required value that provides more information about the machine.

Bits 0-3 = 0x1 for a hardware or microcode machine

0xE for a customer-programmable machine

Bits 4-7 = 0x1 to indicate an IBM product.

0x9 to indicate a non-IBM product.

Model Number:
This optional field can be used to further identify the machine. Possible values include all EBCDIC alphanumeric characters, right-justified and padded with 0x40 characters. If this value is not used or unknown, it is set to zero.

Plant of Manufacture:
This optional field can be used to identify where the machine was produced. Possible values include all EBCDIC alphanumeric characters, right-justified. If this value is not used or unknown, it is set to zero.

Serial Number:
This optional field can be used to further identify the machine. Possible values include all EBCDIC alphanumeric characters, right-justified and padded with 0xF0 characters. If this value is not used or unknown, it is set to zero.

Software Release Level:
This optional field can be used to further identify the level of software used on the machine. Possible values include all EBCDIC alphanumeric characters, right-justified and padded with 0xF0 characters. If this value is not used or unknown, it is set to zero.

Engineering Change Level:
Bytes 52 through 67 can optionally be used to keep track of engineering change levels to the adapter card, control unit, and so on. Possible values include all EBCDIC alphanumeric characters, with user-defined justification and padding. If this field is not used or unknown, it is set to zero.

Error Log

The error log for the DFT device driver is similar in format to that shown in Figure 5-13 on page 5-85. The values for the DFT device driver are defined as follows:

Length: The length of the DFT error log is 4 words.

Class: 0x01 (hardware error)

Subclass: 0x08 (communications adapter)

Mask: 0x0F

Type: 0x04

Error Data Length:
0x0002

Error Data: The DFT device driver uses this area to indicate the probable cause of the error.

LLC Process-To-DFT Device Driver Interface

The DFT device driver uses the LLC-to-device driver interface (the **Send Command SVC** and **Start I/O SVC** for OS LLCs and the **_enque** routine with Start I/O or Send Command queue elements for VRM LLCs) described in "LLC Process-to-Device Driver Interface" on page 4-12. Some additional DFT-specific information and unique device options are defined in the following sections.

Start Device

The **Start device** command is described in "Start Device" on page 4-13 and uses the input queue element shown there. The command extension field for the input queue element is similar to the extension shown there. However, the DFT device driver defines the second word of the extension as follows:

- Byte offset 4 = network ID

  This field is not used by the DFT driver, but it indicates the logical link session that is associated with a particular device ring queue.
- Byte offset 5 = reserved
- Byte offset 6 = IODN of the DFT device driver.

The **Start device** acknowledgment queue element and returned PSB information are described in "Start Device" on page 4-13. Note that this command does not use the device driver correlator field. The first word of the returned command extension contains the address of the device ring queue. This is the only field of the returned command extension used by the DFT device driver.

Halt Device

The **Halt Device** command ends a session with the device driver. This interface and the queue elements used are defined in "Halt Device" on page 4-16. Note that the DFT device driver does not use the device driver correlator field of the input queue element.

Write

The **write** command is issued by an LLC that wishes to send data. Three forms are available, depending on the amount of data to be sent. See "Write Long" on page 4-18, "Write Short" on page 4-20, and "Write (Start I/O)" on page 4-21 for interface definitions. Note that VRM LLCs do not use the interrupt level/sublevel field or the device driver correlator field in any of the queue elements.

For the **Write long** and **Start I/O write** forms of this command, the maximum amount of data that can be sent in a single buffer is 32,768 bytes. If the data does not begin on a 2K-byte boundary, the maximum amount of data that can be sent is reduced by the number of bytes from the 2K-byte boundary to the start of the data. If the amount of data sent exceeds the size of the receive buffer, data will be lost.

Send Status

The **Send status** command is used when the device driver must relay status information for a session to the remote system. The format of the input queue element used by the device driver for this command is similar to that shown in Figure 4-10 on page 4-16. For the **Send status** command, however, the halfword at byte offset 20 (bytes 20 and 21) of the queue element contain status description information. The following values are defined for this field:

0x0000: Data acknowledged

This value indicates that the data or command previously received was valid and the proper response can be made to the host.

0x0001: Read successful

This value indicates that a **read** command received from the host was valid and that a **write** command will follow.

0x0002: Reset

This value indicates that the 3270 reset key has been pressed. The last **write** command should be recalled (if possible).

0x0107: Device not selected

This value indicates a command was received for a device that was not selected.

0x0109: Buffer error

This value indicates that a bad buffer order or a an invalid buffer address was received.

0x0110: Bad command

This value indicates that a non-3270 command was received.

0x0111: Unsupported command

This value indicates that an unsupported 3270 command was received.

The format of the acknowledgment queue element is similar to that shown in Figure 4-11 on page 4-17.

Block I/O Device Ring Queue

This device driver implements a device ring queue as defined in "Device Ring Queue Array" on page 4-34.

Data Buffer Structure

All buffers used by the DFT device driver must be the same length, must begin on a fullword boundary, and must be a multiple of fullwords in length. The data buffer structure is similar to that shown in Figure 4-27 on page 4-36. The DFT device driver defines the first byte of the User Control Area (byte offset 12 of the data buffer structure) as a receive status byte, which indicates BSC receive status. The remaining 63 bytes of the control area are not used by this device driver.

Receive Data

In order to receive data, the DFT device driver takes a buffer from the SLIH ring queue and waits for an interrupt from the adapter. The interrupt indicates that data is available to be received from the card. The driver then places the data into the buffer and generates an interrupt when complete. The driver then places the buffer's segment offset into the device ring queue. If the device ring queue was empty, the LLC is notified (by a **_post** for VRM LLCs or a virtual interrupt for OS LLCs) that a buffer of data has been received. If the device ring queue was not empty, the LLC is expected to process all the buffers in the device ring queue until the queue is empty. See Figure 4-27 on page 4-36 for the format of the data buffer. The LLC must check the Data Offset field to determine the actual start of buffer data.

The driver increments the Buffers Used count each time a buffer is obtained from the SLIH ring. The number of buffers used is compared to the SLIH Threshold value of the DDS. When the Buffers Used value equals the SLIH Threshold value, the driver does a **_post** to the device manager to obtain more pinned receive buffers.

When the SLIH ring queue is empty and the driver tries to obtain a receive buffer, or when the device ring queue is full when the driver tries to add a receive buffer to it, the LLC process is notified by way of a virtual interrupt.

Operation Options

The operation options field for input queue elements used by the DFT device driver is defined in "Operation Options" on page 4-26.

Operation Results

Operation results for a command or unsolicited interrupt are defined in "Operation Results" on page 4-42.

The acknowledgment by the device driver to an LLC for the **start device, halt device, write**, and **send status** commands are always solicited. These operation results are defined as follows:

0x0000: Successful operation.
0x8200: Open time out, control unit down.
0x8300: Adapter not present.
0x8301: Invalid DDS passed.
0x8302: Device driver already opened.
0x8303: Unsupported SNA control unit.
0x8304: No session established with control unit.
0x8305: Host session not established.
0x8306: Unsupported control unit.
| 0x8307: Command rejected due to host contention.
0x8601: No session available - NetID table full.
0x8800: Invalid queue element.
0x8801: Invalid command.
0x8802: Invalid device driver correlator.
| 0x8903: Device not open yet.
| 0x8905: Control unit down.

If there is receive data for an operating system LLC or if an error occurs that is not related to a specific command, the device driver may send an unsolicited queue element (virtual interrupt) to the LLC. This queue element is similar in format to Figure 4-17 on page 4-23 and is defined as follows:

Path ID: ID of the path connecting the operating system LLC to the DFT device driver.
Type: Set to zero for acknowledgment queue element.
Flags: Set to 0x00 to indicate an unsolicited queue element.
Operation Results:
Possible values for this halfword are defined as follows:

0x2xxx = Check status field valid. This is a mask value for unsolicited PSBs.

0x0100 = Receive data available for an OS LLC.

0x8200 = Timeout error.

0x8400 = SLIH ring queue underflow.

0x8500 = Device ring queue underflow.

0x8700 = Buffer pool depleted.

0x0900 = Device selected - indicates the host has locked the device.

0x0A00 = Selection complete - indicates the host has unlocked the device.

IODN: IODN of the 3278/79 Emulation Advanced Adapter port.

Reserved field:
Byte offsets 16 through 23 are reserved.

Check status:
Byte offset 24 contains a halfword value indicating status.

This field is defined as follows:

Bit 0: When set, indicates a fatal link error has occurred and all sessions should be halted.

Bit 1: When set, indicates a communication check has occurred.

Bit 2: When set, indicates a program check has occurred.

Bit 3: When set, indicates a machine check has occurred.

Bits 4-7: Contain a 3270 error indicator in packed decimal format.

Bits 8-11: Contain a 3270 error indicator in packed decimal format.

Bits 12-15: Contain a 3270 error indicator in packed decimal format.

LLC Ring Correlator:
Correlator value that was supplied with the **start** command.

Interrupt Level/Sublevel:
Indicates the level and sublevel on which to send the virtual interrupt to the virtual machine. Bit 0 is set equal to 1, bits 1-15 are reserved, bits 16-23 contain the level, and bits 24-31 contain the sublevel.

Multiprotocol Device Driver

The multiprotocol device driver is a component of the block I/O subsystem. The device driver controls the IBM Multiprotocol Communications Adapter. Each adapter provides three physical communications ports (two RS-232C ports and one X.21 port). The driver supports two ports per adapter and two adapters per machine, for a total of four possible multiprotocol ports.

This device driver supports only half-duplex data transmissions.

The multiprotocol adapter uses alternate direct memory access (DMA) on DMA channels 1 or 5. If other adapters use the same DMA channels as the multiprotocol adapter (or if two multiprotocol adapters on the same machine attempt to use the same DMA channel), the results are unpredictable. See *VRM Programming Reference* for more information on the use of DMA on the RT PC.

For additional information on the block I/O subsystem and block I/O device drivers, see "Block I/O Device Driver Considerations" on page 4-5 and "Block I/O Device Manager" on page 4-38.

The multiprotocol device driver is so named because it can handle any of the following communications protocols implemented by an LLC process:

- Binary Synchronous Communication (BSC)
- Synchronous Data Link Control (SDLC).

The driver also supports the Asynchronous Communication protocol for connecting to an auto-dial modem.

Data transmission rates for the adapter depend on the type of port and the protocol in use. The following rates are available:

	Maximum Data Rate	
Protocol	**RS-232C**	**X.21**
BSC	9,600 bps	19,200 bps
SDLC	19,200 bps	64,000 bps

Figure 4-40. Multiprotocol Data Transmission Rates

For DTE clocking, the supported data rates are 600 and 1,200 baud. For DCE clocking (RS-232C), supported data rates are 300 to 19,200 baud.

Define Device Header

The define device structure (DDS) for the multiprotocol device driver is created at configuration time and passed to the driver by the **Define Device SVC**. Each port must have its own DDS.

Figure 2-2 on page 2-5 shows the format of a DDS.

Pertinent values for the multiprotocol device driver are defined as follows:

IODN: This field contains a number that identifies the device for I/O activity. The multiprotocol device driver does not contain code that is dependent on a particular value in this field.

IOCN: This field contains a number that links the device to the code that supports it.

Define options: This field contains flags that indicate whether the device is being added to or deleted from the VRM.

Device type: 0x0003

Define name: This field is ignored by the VRM, but it can be used to create convention names that identify the device driver.

Offset to hardware characteristics: 28 bytes

Offset to device characteristics: 72 bytes

Offset to error log: 104 bytes.

Hardware Characteristics

The format of this field is shown in Figure 2-3 on page 2-6. Values for the multiprotocol device driver are defined as follows:

Length: 11 words

Internal device type:

0x91xx4000 - indicates the first port.
0x91xx4001 - indicates the second port.
(where xx is set to the slot occupied by the adapter).

The device width field is initialized to 8 bits, but DMA transfers are 16 bits wide.

I/O port address (base): 0x00000510 or 0x00000910, depending on how the adapter is jumpered.

I/O port addresses (number): Set to 0x00000020.

Bus memory start address (RAM): Not used (0).

Bus memory end address (RAM): Not used (0).

DMA type:

- DMA supported
- Alternate DMA
- Only use DMA
- VRM does not enable DMA
- No scatter gather support
- Number of windows field = 0010 (log of 4, which is the number of windows supported)
- Channel numbers:
  - 01 for channel 1
  - 05 for channel 5

Interrupt type:

- Interrupts enabled and shareable
- Class = 2
- Interrupt levels:
  - 0x0A
  - 0x0B

    Do not use the same interrupt level as that used by asynchronous adapters or unbuffered devices.

Device Characteristics

Figure 4-21 on page 4-29 shows the device characteristics fields for a block I/O device driver.

The values for the multiprotocol device driver are defined as follows:

Length of Device Characteristics Section: 8 words

Communications Area Segment ID: The value is placed in the DDS by the configuration process after construction of the communications area.

Communications Area Address: The value is placed in the DDS by the configuration process after construction of the communications area.

Buffer Size: This is a user-specified value placed in the DDS by the configuration process that gives the length of the individual buffers in the buffer pool. This value must be a multiple of 4 so buffers will hold fullwords of data.

Number of Buffers: This is a user-specified value placed in the DDS by the configuration process that gives the number of buffers in the buffer pool.

Device Ring Buffer: This is a user-specified value placed in the DDS by the configuration process that gives the number of buffers to be put on each device ring queue.

SLIH Ring Buffers: This is a user-specified value placed in the DDS by the configuration process that gives the number of receive buffers to be put on the driver's SLIH ring queue.

Maximum # LLCs: This is set to one for the multiprotocol device driver.

NetID Length: The length, in bytes, of the device driver network ID. The multiprotocol device driver uses a null NetID, so this field is set to 0.

NetID Pointer: Set to zero.

Maximum # NetIDs: This value is initially set to 1, but may be set to other values to obtain additional ring queues for use by the LLC process.

SLIH Threshold: This is a user-specified value placed in the DDS by the configuration process that indicates the number of SLIH ring queue buffers that the device driver can use before requesting additional buffers from the device manager.

SLIH Count: The number of SLIH ring queue buffers that the device driver used. It is initially set to 0 and is incremented each time a buffer is moved from the SLIH ring queue to a device ring queue. When the value reaches the SLIH threshold, the device driver resets the field to 0 and issues a **_post** to the device manager.

Microcode IOCN: This value is assigned when the configuration process installs the microcode module with the **Define Code SVC**. The IOCN value is then placed into that port's DDS. The first word of the microcode module consists of a 32-bit value that indicates the length of the module. The rest of the module is the IPL microcode itself.

SLIH Ring Address: This is the address of the driver's SLIH ring queue. It is filled in by the device driver on an **open** command. The segment ID of the SLIH ring queue is the same as the DDS.

Error Log

The format of the error log of the multiprotocol device driver is shown in the following figure:

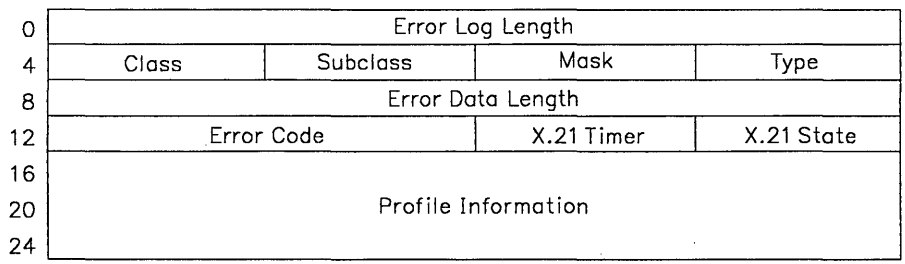


Figure 4-41. Multiprotocol Error Log Format

The fields in the preceding figure are defined as follows:

Error Log Length:
Set to 7 words

Class: Set to 0x01 for hardware error

Subclass: Set to 0x08 for communications adapter

Mask: Set to 0x08 for an RS-232C error or 0x09 for an X.21 error

Type: Set to 0x80 to indicate a permanent error

Error Data Length:
5 words

Error Code: Indicates the type of error logged. Possible values include:

0x8201 = Transmit failsafe timeout.

0x8202 = Inactivity timeout.

0x8280 = X.21 timeout during call establishment.

0x82C0 = Data set ready timeout during call establishment.

0x82C1 = Data set ready did not go off during call termination.

0x8300 = Adapter not present or not functional.

0x8301 = Adapter microcode checksum error.

0x8302 = Adapter microcode IPL timeout.

0x8400 = SLIH ring queue underflow.

0x8401 = Receive overrun, receive DMA not set up.

0x8500 = Device ring queue overflow.

0x8601 = Transmit underrun.

0x8602 = Clear to send timeout during transmit.

0x8900 = Invalid interrupt.

0x8901 = Clear to send dropped during transmit.

0x8903 = Unexpected clear received from data communications equipment for X.21.

0x8904 = Data set ready on for a switched line before data terminal ready.

X.21 Timer: For 0x8280 errors (X.21 timeout), this field indicates the timer that timed out. If the error logged is not 0x8280, this field is reserved. Possible values include:

0x20 = DCE timer (DCE did not go Ready)

0x23 = T1 - DCE did not present Proceed to Select

0x24 = T2 - DCE did not respond after selection signals sent

0x25 = T3B - DCE did not respond after CPS received

0x26 = T4B - DCE did not respond after incoming call accepted

0x27 = T5 - DCE did not respond to DTE clear request

0x28 = T6 - DCE did not respond to DTE clear confirmation.

X.21 State: This field indicates the X.21 process state when an error occurs during X.21 call establishment.

Profile Information:
This field contains a copy of the configuration profile (bytes 4-15 of the command extension) passed with the **Start device** command. See Figure 4-42 on page 4-75 for the format of this profile.

LLC Process-To-Multiprotocol Device Driver Interface

The multiprotocol device driver uses the LLC-to-device driver interface (the **Send Command SVC** and **Start I/O SVC** for OS LLCs and the **_enque** routine with Start I/O or Send Command queue elements for VRM LLCs) described in "LLC Process-to-Device Driver Interface" on page 4-12. Some additional multiprotocol-specific information and unique device options are defined in the following sections.

Start Device

The **Start device** command is described in "Start Device" on page 4-13 and uses the input queue element shown there. The command extension field used by the multiprotocol device driver, which includes configuration information necessary to establish a connection, is shown in Figure 4-42 on page 4-75.

A **Start device** command is issued so that a port can receive data. When a **Start device** completes successfully, a receive timer is usually started in anticipation of the receive data. The timer value is defined in the command extension.

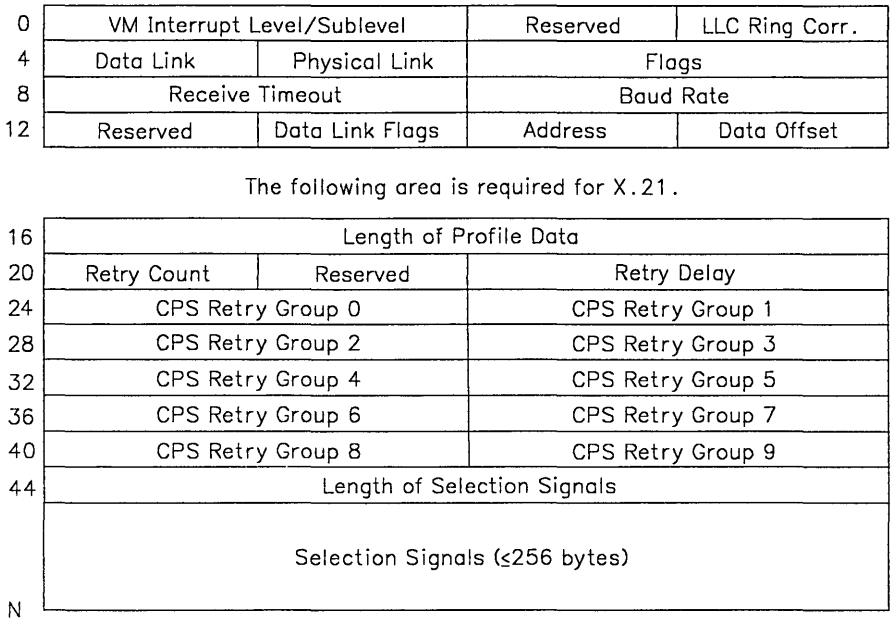


Figure 4-42. Command Extension for the Start Device Command. Bytes 4 through 15 of the command extension contain the multiprotocol configuration profile and are present for all supported protocols. Some additional fields are appended to the header for X.21 and auto-dial use.

The fields in the preceding figure are defined as follows:

VM Interrupt Level/Sublevel:
This is the level and sublevel on which virtual interrupts are returned to the OS LLC process. For VRM LLCs, this field is set to zero.

LLC Ring Correlator:
Byte value assigned by the LLC Process. All queue elements returned for the specified port contain the LLC ring correlator as a return parameter.

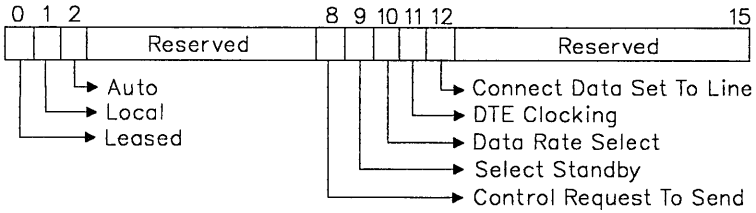
Data Link: This field indicates the data link protocol. Possible values include:

0 = SDLC
2 = BSC

Physical Link:
This field indicates the physical link protocol. Possible values include:

0 = RS-232C
2 = X.21
5 = auto-dial modem

Flags: This halfword contains a bit mask of information needed to establish a connection. The bits are shown in the following figure and defined in the section that follows:



- Leased — Bit 0 indicates whether the telephone circuit is switched (bit set to 0) or leased (bit set to 1).
- Local — Bit 1 indicates whether the call is incoming (bit set to 0) or outgoing (bit set to 1).
- Auto — Bit 2 indicates whether the call must be answered or dialed manually by an operator (bit set to 0) or whether the call can be answered or dialed automatically (bit set to 1).
- Bits 3 through 7 are reserved.
- Control Request to Send — Bit 8 indicates how to manage Request to Send (RTS). When bit 8 equals 0, RTS is disabled after each transmission; when bit 8 equals 1, RTS is enabled all the time.
- Select Standby — Bit 9 indicates, for modems that have the capability to operate on both a primary (leased non-voice grade line) or standby (switched voice grade line) basis. When bit 9 equals 0, standby is not selected; when bit 9 equals 1, standby is selected. For modems that do not have this capability, this bit is set to 0.
- Data Rate Select — Bit 10 indicates, for modems that support a half or alternate transmission speed, whether the alternate speed is selected. When bit 10 equals 0, data rate select is not enabled; when bit 10 equals 1, data rate select is enabled. For modems that do not have this capability, this bit is set to 0.
- DTE Clocking — Bit 11 indicates whether DTE clocking is selected. When bit 11 equals 0, DTE clocking is not enabled; when bit 11 equals 1, DTE clocking is enabled, and the baud rate is indicated in the baud rate field of the command extension.
- Connect Data Set to Line — Bit 12 indicates when DTR is enabled. When bit 12 equals 0, DTR is enabled without waiting for a ring indicate (RI); when bit 12 equals 1, DTR is enabled after an RI. Note that if DSR goes active prior to RI, DTR is enabled regardless of RI.
- Bits 13 through 15 are reserved.

Receive Timeout:

This field indicates whether a receive timer is started in anticipation of receive data after the **Start device** command completes successfully or a transmit final is detected. If this field equals zero, no receive timer is started. Non-zero values indicate the duration of the timer in 100 millisecond intervals. If a receive occurs that is not a receive final,

the timer is restarted until a receive final occurs. If the timer expires prior to a receive, a receive timeout is reported to the LLC process and an inactivity timer is started. The receive timeout value can be changed by a **Change parameters** command after a **Start device** is completed.

Baud Rate: When DTE clocking is selected from bit 11 of the flags field, the baud rate field indicates the data rate. The following values are defined for this field:

0x0258 = 600 bps
0x04B0 = 1200 bps
0x07D0 = 2,000 bps
0x0960 = 2,400 bps

Data Link Flags:

Bits 0, 1, and 7 of this byte are used as flags to further define the data link characteristics for the various protocols. Bits 2 through 6 are reserved.

For SDLC, bit 0 indicates the type of encoding to be used. When bit 0 equals 0, the normal non-return to zero encoding and decoding (NRZ) is used; when bit 0 equals 1, non-return to zero inverted encoding and decoding (NRZI) is used. Bit 1 indicates whether selective addressing is to be enabled. When bit 1 equals 0, selective addressing is not enabled; when bit 1 equals 1, selective addressing is enabled and the specified address is located in the Address field of the command extension. Bit 7 indicates whether the receive timer is restarted after a receive timeout. When bit 7 equals 0, the timer is not restarted. When bit 7 equals 1, the receive timer is restarted after a receive timeout.

For BSC protocol, bit 0 indicates whether encoding is done in EBCDIC (bit 0 equals 0) or in ASCII (bit 0 equals 1) and, therefore, whether the block check character is computed by cyclic redundancy check or longitudinal redundancy check. For ASCII encoding, odd parity checking and generation is performed on each data byte. Bit 1 indicates whether control unit address mode is enabled. When bit 1 equals 0, address compare mode is not enabled; when bit 1 equals 1, address compare mode is enabled. If enabled, the address to be used is contained in the Address field of the command extension. Bit 7 indicates whether the receive timer is restarted after a receive timeout. When bit 7 equals 0, the timer is not restarted. When bit 7 equals 1, the receive timer is restarted after a receive timeout.

Address: This field contains the address to use for selective addressing (SDLC) or address comparison (BSC) when bit 1 of the Data Link Flags field is set to 1. If bit 1 of the Data Link Flags field is 0, the Address field is ignored.

Data Offset:

This field indicates an offset into the data area of a receive buffer where the multiprotocol device driver is to begin placing data. This field can be used by the LLC process to force word alignment of specific fields in the receive data stream.

The following fields are appended to the command extension for X.21. For auto-dial, byte offset 16 contains the length of the dial data. The dial data begins at byte offset 20 and can be up to 80 bytes in length. The following fields are defined for X.21:

Length of Profile Data:
This field indicates the total length of the X.21 profile area, including the selection signals.

Retry Count:
For outgoing calls, this field determines the number of retries made by the device driver to establish the call before the call is considered to have failed. Possible values for this field range from 0x00 to 0x0F, and the default number of retries is 0x08. Note that some combinations of retry count and retry delay values could produce a retry sequence that takes more than 30 minutes to determine a failed call and notify the LLC process. The total time of a retry sequence is important to software components that use a timer with this SVC.

For incoming call attempts, this field indicates the number of times the device driver enters the ready state after a Clear To Send occurs.

Retry Delay:
For outgoing calls, this field determines the number of 100 millisecond intervals between call retries. The default value is 300 intervals (30 seconds). If the LLC process issues an **open** command to listen for an incoming call, this field is ignored.

CPS Retry Group 0-9:
These fields indicate whether a retry should be attempted when a particular Call Progress Signal (CPS) is received.

Each halfword (0-9) reflects status for the corresponding CPS group 0-9. Bits 0-9 of each halfword indicate status and error conditions within each group. Bits 10-15 are reserved. If any of bits 0-9 are set, the call should be retried the number of times indicated in the Retry Count field. If the Retry Count is reached, or is set to zero, the driver notifies the LLC process that the call failed. For more information on the CPS groups and the values defined within each group, see *CCITT Recommendation for X.21, Version 24.*

If more than one CPS is received at the same time, the last one received will be considered significant and determines the prescribed course of action. For Retry Group 0, if the call is not complete after 60 seconds, no retry will be attempted.

IBM recommends that no retries be attempted for received Call Progress Signals that are undefined by X.21 or the network. The LLC, therefore, is notified of all received CPS.

Only outgoing calls can receive a CPS, so this field is not significant for incoming calls.

Length of Selection Signals:
This field indicates the length of the Selection Signal field. The maximum length in bytes is 256.

Selection Signals:
This field contains the Selection Signals, stored here in International Alphabet 5 format.

The **Start device** acknowledgment queue element and returned PSB information are described in "Start Device" on page 4-13. Note that the device driver correlator field is not returned by the

multiprotocol device driver. The returned command extension contains the address of the device ring queue in the first word, followed by a copy of the configuration profile section of the command extension described in Figure 4-42 on page 4-75.

Halt Device

The **Halt Device** command ends a session with the device driver. This interface and the queue elements used are defined in "Halt Device" on page 4-16. Note that this device driver does not use the device driver correlator field of the input queue element.

Write

The **write** command is issued by an LLC that wishes to send data. Three forms are available, depending on the amount of data to be sent. See "Write Long" on page 4-18, "Write Short" on page 4-20, and "Write (Start I/O)" on page 4-21 for interface definitions. Note that the multiprotocol device driver does not use the device driver correlator field of the input queue element.

For the **Write long** and **Start I/O write** forms of this command, the maximum amount of data that can be sent in a single buffer is 32,768 bytes. If the data does not begin on a 2K-byte boundary, the maximum amount of data that can be sent is reduced by the number of bytes from the 2K-byte boundary to the start of the data. If the amount of data sent exceeds the size of the receive buffer, data will be lost.

Start List

The **Start list** command passes an SDLC automatic poll/response control block to the multiprotocol device driver. The control block consists primarily of a list of stations to be processed and is shown in Figure 4-43.

Two modes of list processing are available: auto-poll mode and auto-response mode. The specifics of each mode are described in "List Processing Mode Descriptions" on page 4-80.

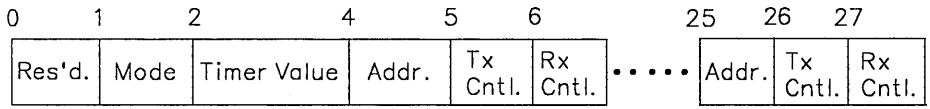


Figure 4-43. Automatic Poll/Response Control Block. Each control block is 28 bytes in length, including 8 required address entries. Address values of 0x00 indicate list items to be skipped.

The fields in the preceding figure are defined as follows:

Mode — This field indicates the type of list processing to perform. A value of 0x00 indicates auto-poll mode, and a value of 0x01 indicates auto-response mode.

Timer Value — This field contains the receive timeout value, expressed in increments of 5 milliseconds. The timer determines how long to wait for a response from remote stations. If the interval expires without response, a buffer is returned to the LLC indicating this

condition. If the timer expires in auto-response mode, the mode is terminated. However, both the adapter and device driver will continue to receive data.

Address — This 1-byte address field identifies the stations in the list to be processed. This field, together with the Tx Cntl. and Rx Cntl. fields, identifies the 8 required entries in the control block. An address value of 0x00 indicates an entry to be skipped (not processed).

Tx Cntl — This field contains the transmit control byte for each station.

Rx Cntl — This field contains the value to be compared against a received control byte.

When the **Start list** command is issued, the device driver takes the received control block, passes it on to the adapter, and returns command completion status to the LLC. Because the command completion information indicates only that the adapter received the control block, IBM recommends that the LLC not request an acknowledgment (bit 0 of the operation options halfword set to 0).

As soon as the adapter receives the control block, the adapter starts processing the list items. Valid entries are processed by first transmitting a frame (poll frame for auto-poll mode or response frame for auto-response mode), turning the line around, and checking the received control byte against the compare value. If the compare bytes are equal, the list entry is completed and processing of the next valid list entry begins.

The **Start list** command uses Send Command queue elements of the same format as that shown in Figure 4-12 on page 4-18. The command extension address field is the address of the automatic poll/response control block, and the command extension length is always 28. The device option for a **Start list** command is 9. The multiprotocol device driver does not use the device driver correlator field.

The acknowledgment queue element for this command has the same format as that shown in Figure 4-13 on page 4-19.

List Processing Mode Descriptions

The multiprotocol device driver supports two modes of processing lists, auto-poll mode and auto-response mode.

In auto-poll mode, each station in the list is sequentially polled. The primary station sends out a poll, starts a timer, and turns the line around to await a response. If the response control byte equals the receive control compare byte, no data is transmitted and processing proceeds to the next station in the list. If the response control byte does not match the response byte, the frame (including data bytes that follow the control byte), is passed to the LLC process by way of a device ring queue buffer. All subsequent frames are also passed to the LLC until a frame with the poll final bit set is encountered. Processing then proceeds to the next frame.

If a timeout or a transmit error occurs, a device queue ring buffer is also passed to the LLC. The receive status byte in the buffer informs the LLC of the error condition, and processing resumes with the next station in the list.

Stations in the list with an address of 0x00 are not polled, and processing proceeds to the next entry.

Auto-poll mode stops only when the last item in the list is processed. When this occurs, a buffer is passed to the LLC with a receive status byte value indicating end of list.

The auto-response mode is used by a secondary station which also passes an 8-item poll list to the device driver, but only one list item is valid. The valid entry is processed by transmitting a response frame, then waiting for a frame to be received. When a frame is received, the control byte is compared and, if it matches the compare value, the specified response is transmitted. A timer is then started and the line is turned around.

In auto-response mode, the list is processed repeatedly, so when the end of the list is reached, processing continues from the top of the list.

A timeout, transmit error, or miscompare for auto-response mode is handled the same way as for auto-poll mode, except that auto-response mode is also terminated. In addition, auto-response mode is terminated if the LLC issues a **Stop list** command. This condition also returns a buffer, and the receive status byte indicates an end-of-list condition.

Stop List

The **Stop list** command is used to terminate list processing for auto-response mode. As with the **Start list** command, the device driver relays the command to the adapter and returns status to the LLC. The acknowledgment to this command means only that the adapter received the command, so IBM recommends that you do not request acknowledgment on completion.

When list processing is terminated as a result of this command, a buffer is returned to the LLC process. The receive status byte of the buffer indicates the end-of-list condition.

The **Stop list** command uses Send Command queue elements of the format shown in Figure 4-10 on page 4-16. The command extension bit is set to zero, and the device option is 10 (decimal).

The acknowledgment queue element has the format shown in Figure 4-11 on page 4-17. Note that the device driver correlator field is not used.

Change Parameters

The **Change parameters** command is used by the LLC to change the receive timeout value in the configuration profile header (shown in Figure 4-42 on page 4-75).

The receive timeout value indicates the number of 100 millisecond increments that can elapse before the LLC is notified of a receive timeout error.

The device driver starts the receive timer when the **Start device** command is issued or upon receipt of a transmit final. The timer is restarted at each receive until a receive final is detected.

A receive timeout value of 0 indicates that the receive timer is not to be run. In this case, no receive timeout errors will be sent to the LLC.

Figure 4-44 on page 4-82 shows the Send Command queue element for the **Change parameters** command.

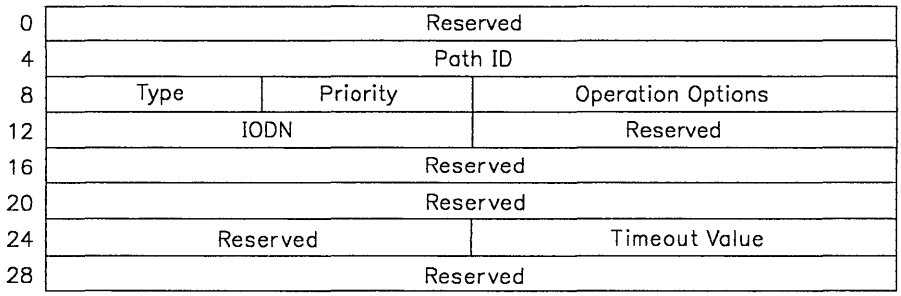


Figure 4-44. Change Parameters Queue Element

The fields in the preceding figure are defined as follows:

Path ID The ID of the LLC-to-device driver path.

Type Set to 1 for a Send Command queue element.

Priority Set to 0.

Operation Options The command extension bit must be set to 0, and the device option is 11 (decimal).

IODN The IODN of the port.

Timeout Value Indicates the number of 100 millisecond increments to be measured before the LLC is notified of a receive timeout. A value of zero indicates that the receive timer is no longer to be run.

The format of the acknowledgment queue element is similar to that shown in Figure 4-11 on page 4-17.

Block I/O Device Ring Queue

This device driver implements a device ring queue as defined in "Device Ring Queue Array" on page 4-34. Because the multiprotocol device driver supports only one network ID at a time, the number of rings field in the device ring queue array is equal to 1.

Data Buffer Structure

All buffers used by the multiprotocol device driver must be the same length, must begin on a fullword boundary, and must be a multiple of fullwords in length. The data buffer structure is similar to that shown in Figure 4-27 on page 4-36. When bit 15 of the flag field is set, it indicates a buffer overflow occurred, and the overflowed data bytes are lost. In addition, the first byte of the User Control Area (byte offset 12 of the data buffer structure) is used to indicate receive status for the various protocols. The rest of the control area is not used by this device driver.

The receive status byte of the User Control Area is defined for the supported protocols in the following sections.

SDLC Receive Status

For the SDLC protocols, the first bit of the receive status byte is reserved, bits 1-3 indicate an index into the auto poll/response station address list (these bits are set to zero if the auto poll/response mode is not in effect), and bits 4-7 indicate receive errors or status conditions detected by the multiprotocol adapter. Errors are prioritized low (0) to high (15), so if multiple errors occur on one frame only the highest priority error is sent. The following hexadecimal error values are defined for bits 4-7 of the receive status byte:

0 = No error or status.

1 = End of poll list status.

This status value indicates that the adapter has completed the auto/poll response list.

2 = Poll list control mismatch.

This error value indicates that the adapter has received a frame and the frame's control byte did not match the control byte expected. The frame is still passed in the buffer, and the adapter processes subsequent frames until a final frame or a receive timeout occurs. For auto-response mode, processing is then terminated; for auto-poll mode, processing continues with the next entry.

3 = Unexpected response address.

This error value, valid only in auto-poll mode, indicates that the adapter received a frame address that did not match an address in the auto-poll list. The adapter processes subsequent frames until a final frame or receive timeout occurs. Processing then continues with the next entry.

4 = Insufficient space in DMA receive buffer.

This error value indicates that received data was lost because the DMA receive buffer was not large enough to contain the current frame. This condition is also noted in the data buffer's flag field. The adapter processes subsequent frames until a final frame or a receive timeout occurs. For auto-response mode, processing is then terminated; auto-poll mode, processing continues with the next entry.

5 = Poll list receive timeout.

This error value indicates that the adapter polled a station that did not respond in time, or that the station was not polled in the expected amount of time. For auto-response mode, processing is then terminated; for auto-poll mode, processing continues with the next entry.

6 = Invalid frame synchronization

The adapter detected an apparent start of a valid frame. As the data progressed, the adapter determined that the frame was not valid, but some of the data was already DMAed to the system. This status can be caused by noise on the line that looks like the start of a valid SDLC frame.

7 = Poll list compare with data.

This status value indicates that, for auto poll/response mode, the received control byte matched the control byte expected and that information was also sent in the frame. The adapter processes subsequent frames until a final frame or receive timeout occurs. For auto-response mode, processing is then terminated; for auto-poll mode, processing continues with the next entry.

8 = Poll list Clear-To-Send timeout.

This error value indicates that, for auto-poll mode only, the Clear To Send (CTS) signal for a transmission did not become active in the required amount of time.

9 = Poll list transmit failsafe timeout.

This error value indicates that a transmission failed to complete within three seconds from the time the first byte was loaded.

A = Off-byte boundary detected

This error value indicates that a received frame with a good frame check sequence failed to end on a byte boundary. The adapter processes subsequent frames until a final frame or a receive timeout occurs. For auto-response mode, processing is then terminated; for auto-poll mode, processing continues with the next entry.

B = Abort sequence detected.

This status value indicates that the adapter has detected an abort bit pattern in the received frame. The adapter processes subsequent frames until a final frame or a receive timeout occurs. For auto-response mode, processing is then terminated; for auto-poll mode, processing continues with the next entry.

C = Poll list transmit underrun.

This error value indicates, for auto-poll mode only, that the adapter had an underrun condition and stopped transmitting the rest of the frame.

D = Frame check sequence (FCS) error.

This error value indicates that the FCS computed by the adapter for a received frame did not match the FCS include in the frame. The adapter processes subsequent frames until a final

frame or a receive timeout occurs. For auto-response mode, processing is then terminated; for auto-poll mode, processing continues with the next entry.

E = Receive overrun error.

This error value indicates that a receive character was lost due to an overrun condition at the adapter level. The adapter processes subsequent frames until a final frame or a receive timeout occurs. For auto-response mode, processing is then terminated; for auto-poll mode, processing continues with the next entry.

F = Receive terminated due to transmit or CTS dropped (for poll/list processing)

This status value indicates that the adapter terminated a receive operation in order to allow a transmit command to use the half-duplex facilities. In this case, the data buffer will contain some data, but not a complete frame.

BSC/X.21 Receive Status

For the BSC/X.21 protocols, the first bit of the receive status byte is reserved, and bits 1-7 indicate receive errors or status conditions detected by the multiprotocol adapter, or BSC message types. The following hexadecimal values are defined for the receive status byte:

00 = No error or status.

04 = DMA receive buffer not large enough.

This error value indicates that receive data was lost because the DMA receive buffer was not large enough to contain the current frame. This condition is also reflected as a receive buffer overflow in the receive buffer's flag field.

08 = Data synchronization lost.

This error value indicates that two data synchronization characters were not detected within three seconds.

0A = Receive parity error.

This error value indicates that at least one byte of a BSC (ASCII mode only) or X.21 frame was received with bad parity.

0B = BSC ID field too long.

This error value indicates that a BSC frame was received with an ID of more than 15 bytes.

0D = BSC block check code (BCC) error.

This error value indicates that the BCC computed by the adapter did not match the BCC in the received frame. This can either be a BCC error (EBCDIC) or a longitudinal redundancy check error (ASCII).

0E = Receive overrun error.

This error value indicates that a receive character was lost due to overrun conditions at the adapter level.

0F = Receive terminated due to transmit.

This status value indicates that the adapter terminated a receive operation in order to allow a transmit command to use the half-duplex facilities. In this case, the data buffer will contain some data, but not a complete frame.

10 = BSC ACK0 message received.

11 = BSC ACK1 message received.

12 = BSC WACK message received.

13 = BSC NAK message received.

15 = BSC ENQ message received.

17 = BSC EOT message received.

18 = BSC RVI message received.

19 = BSC DISC message received.

2A = BSC SOHITB message received.

2B = BSC SOHETB message received.

2C = BSC SOHSTXITB message received.

2D = BSC SOHSTXETB message received.

2E = BSC SOHSTXETX message received.

3C = BSC STXITB message received.

3D = BSC STXETB message received.

3E = BSC STXETX message received.

3F = BSC STXENQ message received.

4C = BSC SOHXSTXITB message received.

4D = BSC SOHXSTXETB message received.

4E = BSC SOHXSTXETX message received.

5C = BSC XSTXITB message received.

5D = BSC XSTXETB message received.

5E = BSC XSTXETX message received.

60 = BSC data ACK0 message received.

61 = BSC data ACK1 message received.

63 = BSC data NAK message received.

65 = BSC data ENQ message received.

76 = BSC abort message received.

77 = BSC trailing pad not detected.

7F = BSC unrecognizable message received.

ASC Receive Status

For the ASC protocol, the receive status byte is valid only during auto-dial. All receive data is buffered until DSR goes active and is then passed to the LLC process. During auto-dial, bits 0-3 of the receive status byte are reserved, and bits 4-7 indicate receive errors or status conditions detected by the multiprotocol adapter. The following hexadecimal values are defined for bits 4-7 of the receive status byte:

0 = No status or error.

4 = DMA receive buffer not large enough.

This error value indicates that receive data was lost because the DMA receive buffer was not large enough to contain the current frame. This condition is also reflected as a receive buffer overflow in the receive buffer's flag field.

A = Parity error.

This error value indicates that a character was received with incorrect parity.

D = Framing error.

This error value indicates that a receive character was incorrectly framed.

E = Overrun error.

This error value indicates that a receive character was lost due to an overrun occurring on the adapter level.

Receive Data

In order to receive data, the multiprotocol device driver supplies the multiprotocol adapter with a buffer obtained from the SLIH ring queue. When the adapter receives data, the adapter places the data into the buffer with DMA and generates an interrupt when complete. The driver then places the buffer's segment offset into the device ring queue. If the device ring queue was empty, the LLC is notified (by a **_post** for VRM LLCs or a virtual interrupt for OS LLCs) that a buffer of data has been received. If the device ring queue was not empty, the LLC is expected to process all the buffers in the device ring queue until the queue is empty. See Figure 4-27 on page 4-36 for the format of the data buffer. The LLC must check the Data Offset field to determine the actual start of buffer data.

Whenever a receive is completed, the driver must obtain another buffer for the adapter from the SLIH ring queue. The driver increments the Buffers Used count each time a buffer is obtained from

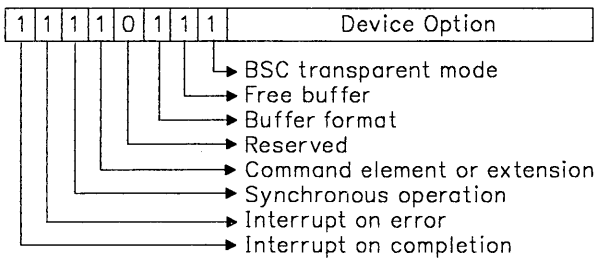
the SLIH ring. The number of buffers used is compared to the SLIH Threshold value of the DDS. When the Buffers Used value equals the SLIH Threshold value, the driver does a **_post** to the device manager to obtain more pinned receive buffers.

If the device ring queue is full when the driver tries to add a receive buffer to it, the LLC process is notified by way of a virtual interrupt.

If the SLIH ring queue is empty when the driver tries to obtain a receive buffer, the driver logs an error but does not notify the LLC.

Operation Options

The operation options field for input queue elements used by the multiprotocol device driver is shown in the following figure. "Operation Options" on page 4-26



The fields in the preceding figure are defined in "Operation Options" on page 4-26. One additional bit is defined for the multiprotocol device driver. The BSC transparent mode bit indicates whether transmissions are made in transparent mode. This bit flag applies to **Write** commands only.

Operation Results

The acknowledgment by the device driver to an LLC for the **start device**, **halt device**, **write**, **start list**, **stop list**, and **change parameters** commands are always solicited. These operation results, as well as other unsolicited operation results, are defined as follows:

0x0000: Successful operation.

0x0100: Receive data available.

0x8200: Receive timeout.

This error occurs when a timer, started in anticipation of receiving data, expires before data is received. When this occurs, the LLC is notified of the timeout and an inactivity timer is started.

0x8201:	Transmit failsafe timeout. This error occurs when a **transmit** command is sent to the adapter and a timer, started in anticipation of a transmit complete signal from the adapter, expires. In this case, the LLC is notified of the timeout and an inactivity timer is started.
0x8202:	Inactivity timeout. This error occurs when a timer, started to ensure the adapter is still functional during periods of inactivity, expires. If an interrupt occurred while the inactivity timer was running, the adapter is known to be functional and the timer is restarted. If no interrupt occurred, the device driver tries to force an adapter interrupt. If the timer again expires after a retry, the LLC is notified that the adapter is inactive.
0x8280:	X.21 timeout during call establishment. This error occurs when the timer used to monitor X.21 call establishment expires. In this case, the LLC is notified of the timeout and sends a DTE clear request.
0x82C0:	Data Set Ready (DSR) timeout. This error occurs if DSR is not indicated within 20 seconds of Data Terminal Ready. In this case, the LLC is notified of the timeout and the inactivity timer is started.
0x82C1:	Data Set Ready failed to go off during call termination.
0x8300:	Adapter not present or not functional.
0x8301:	Adapter microcode checksum error.
0x8302:	Adapter microcode IPL timeout. This error occurs when the adapter fails to notify the LLC of successful IPL within 1 second after receiving the microcode. In this case, the LLC is notified of the IPL timeout.
0x8303:	**Start device** command received when device already started.
0x8304:	Device not open or not started.
0x8305:	Command stopped due to **Halt device** command.
0x8500:	Device ring queue overflow.
0x8601:	Transmit underrun.
0x8602:	Clear To Send (CTS) timeout on transmit.
0x8800:	Invalid queue element type.
0x8801:	Invalid command.
0x8802:	Invalid transmit buffer.
0x8803:	Final block indicated for a **Start I/O write** command before all buffers transmitted.
0x8900:	Invalid interrupt.
0x8901:	CTS dropped during transmit.
0x8902:	DSR dropped.
0x8903:	Unexpected Clear received in X.21.
0x8904:	DSR on before DTR raised for a switched line.
0x8A00:	Auto-dial modem data.
0x8A01:	X.21 call progress signal - call not completed
0x8A02:	X.21 DCE provided information.
0x8A03:	Ready for manual dial.
0x8A04:	X.21 call progress signal - information only
0xCnnn:	Definition is equivalent to 0x8nnn definition in this list.

Token Ring Device Driver

The token ring device driver is a component of the block I/O subsystem.
The driver supports up to four Token Ring adapters. Each adapter has a single port (IODN). Only one virtual machine can use an adapter at a time.

For additional information on block I/O device drivers, see "Block I/O Device Driver Considerations" on page 4-5.

Define Device Header

The define device structure (DDS) for the token ring device driver is created at configuration time and passed to the driver by the **Define Device SVC**. Each active adapter must have its own DDS.

Figure 2-2 on page 2-5 shows the format of a DDS.

Pertinent values for the token ring device driver are defined as follows:

IODN: This field contains a number that identifies the device for I/O activity.

IOCN: This field contains a number that links the device to the code that supports it.

Define options: This field contains an indication of the device definition options.

Device type: 0x0003.

Define name: This field is ignored by the VRM, but it can be used to create convention names that identify the device driver.

Offset to hardware characteristics: 28 bytes.

Offset to device characteristics: 80 bytes.

Offset to error log: 0.

Hardware Characteristics

The format of this field is shown in Figure 2-3 on page 2-6. Values for the token ring device driver are defined as follows:

Length: 11 words.
Internal device type:
0xA1xx30nn

(where xx is set to the slot occupied by the adapter and nn identifies the port number – 00, 01, 02, or 03).

I/O port address (base):
Depends on how the adapter is jumpered. Possible values, along with the corresponding DMA channel, include:

Base Address	DMA Channel
0x000001C0	5
0x00000140	6
0x000011C0	7
0x000011D0	3

I/O port addresses (number):
0x00000010 for 16 addresses
Bus memory start address (RAM):
Not used (0).
Bus memory end address (RAM):
Not used (0).
DMA type:
Set to 0xE20004nn (where nn indicates the channel that corresponds to the base I/O address of the adapter).
Interrupt type:
0x800002xx (where xx is 11 or 12 as set by jumpers on the adapter).
Bus memory start address (ROM):
Not used (0).
Bus memory end address (ROM):
Not used (0).

Device Characteristics

Figure 4-21 on page 4-29 shows the device characteristics fields for a block I/O device driver. The token ring device driver uses two words for device-dependent characteristics. These characteristics are a 6-byte node address and a 2-byte area indicating options for the **open** command.

The values for a token ring device are defined as follows:

Length of Device Characteristics Section:
10 words.

Communications Area Segment ID:
The value is placed in the DDS by the configuration process after construction of the communications area.

Communications Area Address:
The value is placed in the DDS by the configuration process after construction of the communications area.

Buffer Size:
: This is a user-specified value placed in the DDS by the configuration process that gives the length of the individual buffers in the buffer pool. For information on the buffer size, see "Data Buffer Structure" on page 4-50.

Number of Buffers:
: This is a user-specified value placed in the DDS by the configuration process that gives the number of buffers in the buffer pool.

Device Ring Buffer:
: This is a user-specified value placed in the DDS by the configuration process that gives the number of buffers to be put on each device ring queue.

SLIH Ring Buffers:
: This is a user-specified value placed in the DDS by the configuration process that gives the number of receive buffers to be put on the driver's SLIH ring queue.

Maximum # LLCs:
: This is a user-specified value placed in the DDS by the configuration process that gives the maximum number of LLCs that may attach to the device driver at one time.

NetID Length:
: The length, in bytes, of the device driver network ID. This value is fixed at 1 byte.

NetID Pointer:
: This is a user-specified value placed in the DDS by the configuration process that gives the offset, in bytes, to the received packet of the network ID.

Maximum # NetIDs:
: This is a user-specified value placed in the DDS by the configuration process that gives the maximum number of Network IDs that the device driver can support.

Data Offset:
: This value indicates an offset (in bytes) into the data area of the receive buffer at which the device driver is to place received data. Not all device drivers use this field.

SLIH Threshold:
: This is a user-specified value placed in the DDS by the configuration process that indicates the number of SLIH ring queue buffers that the device driver can use before requesting additional buffers from the device manager.

SLIH Count:
: The number of SLIH ring queue buffers that the device driver used. It is initially set to 0 and is incremented each time a buffer is moved from the SLIH ring queue to a device ring queue. When the value reaches the SLIH threshold, the device driver resets the field to 0 and issues a **_post** to the device manager.

Microcode IOCN:
: Set to 0 (not used).

SLIH Ring Address:
This is the address of the driver's SLIH ring queue. It is filled in by the device driver on an **open** command. The segment ID of the SLIH ring queue is the same as the DDS.

Node Address:
The user can specify a node address for the adapter. If no address is provided by the user, the burned-in address of the adapter is the node address. Either way, bit 0 of byte 0 must be set to 0.

Open Options:
This two-byte field defines the options sent to the adapter when the **open** command is issued:

- Bit 0 = wrap interface
- Bit 1 = disable hard interrupts
- Bit 2 = disable soft interrupts
- Bit 3 = pass adapter medium access control (MAC) frames
- Bit 4 = pass attention MAC frames
- Bit 5 = pad routing field
- Bit 6 = frame hold
- Bit 7 = contender
- Bit 8 = pass beacon MAC frame
- Bits 9 through 15 are reserved.

Block I/O Communication Area

Each token ring adapter has a unique communication area associated with it. The format of this area is described in "Block I/O Communication Area" on page 4-33.

Data Buffer Structure

All buffers must begin on a fullword boundary, must be a multiple of words in length, and must be large enough to hold the maximum size packet, the header, and a data offset value as specified in the device characteristics section of the token ring DDS. For more information, see **_bfget** in *VRM Programming Reference*.

The token ring device driver implements this structure as defined in Figure 4-27 on page 4-36 with the exception of the low-order bit of the flag field and the user control area.

The low-order bit of the flag field is defined as the receive buffer overflow and is set to indicate that the buffer is not large enough to hold the entire receive packet. If the receive buffer overflow bit is set, the buffer has accepted as much of the packet as will fit in the buffer. The remainder of the packet has been discarded.

The 64-byte user control is not used by the token ring device driver.

LLC Process-To-Token Ring Device Driver Interface

The token ring device driver uses the LLC-to-device driver interface (the **Send Command SVC** and **Start I/O SVC** for OS LLCs and the **_enque** routine with Start I/O or Send Command queue elements for VRM LLCs) described in "LLC Process-to-Device Driver Interface" on page 4-12. Some additional token ring-specific information, including a token ring-specific device option, is defined in the following section.

Start Device

For OS LLCs that interface to the token ring device driver, a command extension is always used. Bit 19 of GPR2 is therefore set to one. The command extension area is defined as follows:

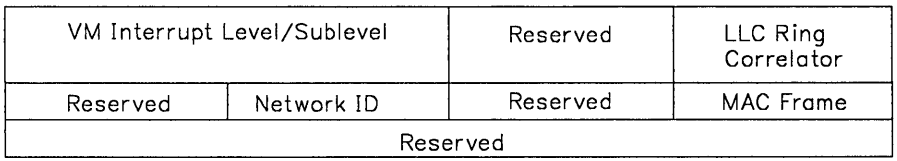


Figure 4-46. Command Extension for the Start Device Command

VM Interrupt Level/Sublevel:
: This is the level and sublevel on which virtual interrupts are returned to the LLC process.

LLC Ring Correlator:
: Byte value assigned by the LLC Process. All virtual interrupts returned for the following network ID contain the LLC ring correlator as a return parameter.

Network ID:
: A 1-byte value that indicates how the LLC identifies itself on the network.

MAC Frame:
: If this **start** command is for MAC frame processing, this field is set to 0x01 and the NetID field must be 0x00. Only one MAC frame **start device** command is allowed per adapter.

The returned PSB information is described in "Start Device" on page 4-13. The format of the returned command extension is shown in the following figure.

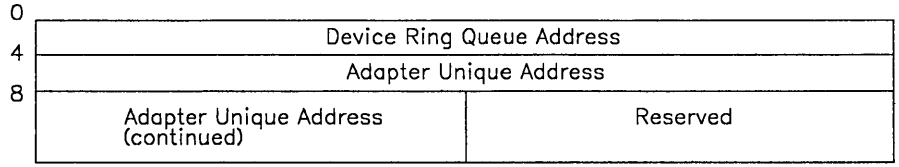


Figure 4-47. Returned Command Extension for the Start Device Command

Device Ring Queue address:
This is the segment offset of the Device Ring Queue assigned to the Network ID passed on the **Start Device** command. The segment ID is the same as for the buffer pool and is not returned.

Adapter unique address:
This 6-byte field contains the address of the adapter. The adapter reads this field and returns it to the LLC for use in the addressing on packets.

For VRM LLCs that interface to the token ring device driver, a command extension is always used. The input queue element, command extension, and acknowledgment queue element are defined in "Start Device" on page 4-13.

Halt Device

The **Halt Device** command ends a session with the device driver. This interface is defined in "Halt Device" on page 4-16.

Write

The **write** command is issued by an LLC that wishes to send data. The token ring device driver supports the **write long** and **start I/O write** forms of this command. The driver uses two transmit buffers on the adapter to send write data. See "Write Long" on page 4-18 and "Write (Start I/O)" on page 4-21 for interface definitions. Level/Sublevel fields in either of the acknowledgment queue elements.

Query Statistics

The **query statistics** command is used by the LLC to read the counter values accumulated by the device driver. The counters are initialized to 0 by the **open** command and are cleared by setting bit 21 of the operation options field.

OS LLCs use the **Send Command SVC** to issue this command. Input parameters are:

- GPR2 = IODN for the desired adapter (bits 0-15), operation options (bits 16-31)
  - Bit 16 = Interrupt on completion (optional)
  - Bit 17 = Interrupt on error (optional)

- Bit 18 = Synchronous operation (optional)
- Bit 19 = Command extension (always 1)
- Bit 20 = Reserved
- Bit 21 = clear counters after **query statistics**
- Bits 22-23 = Reserved
- Bits 24-31 = 9 for query statistics.

- GPR3 = Reserved
- GPR4 = Reserved
- GPR5 = Command extension address
- GPR6 = Command extension length (minimum of 64 bytes)
- GPR7 = ID of the LLC-to-device driver path.

The command extension passes additional information as shown in the following figure.

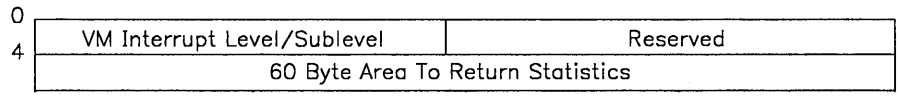


Figure 4-48. Command Extension for Query Statistics Command

Returned information for the SVC is contained in GPR2. Additional information is contained in the PSB and returned command extension.

Program status block completion information includes:

Status flag:	0x14 to indicate a solicited interrupt.
Overrun count:	Set to 0.
Operation results:	Bits 0-15, see "Operation Results" on page 4-42; bits 16-31 contain the IODN for the token ring adapter.
Data word 1:	Operation options (bits 0-15); segment ID of the command extension (bits 16-31).
Data word 2:	Command extension address.
Data word 3:	Reserved (set equal to 0).

The returned command extension area is defined as follows:

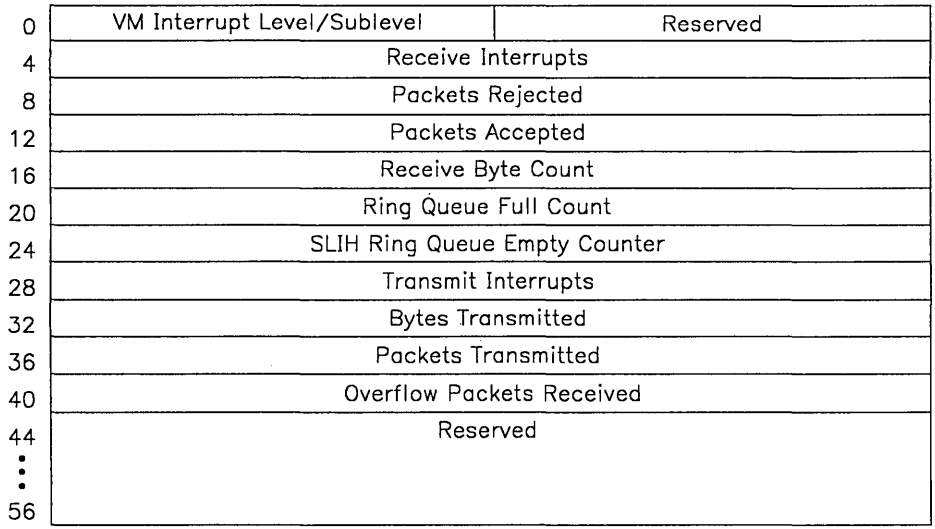


Figure 4-49. Returned Command Extension for the Query Statistics Command

VM Interrupt Level/Sublevel:
The interrupt level and sublevel of the LLC that initiated the **query statistics** command.

Receive Interrupts:
Total number of receive interrupts processed.

Packets Rejected:
Total number of packets rejected, no NetID match.

Packets Accepted:
Total number of packets accepted.

Receive Byte Count:
Total number of bytes received and placed in the receive buffer.

Ring Queue Full Count:
Total number of times any device ring queue went full and a **_post** or unsolicited interrupt was sent to the LLC. A packet is lost each time this occurs.

SLIH Ring Queue Empty Counter:
Total number of times the SLIH ring queue went empty. This indicates that a reconfiguration may be necessary to allocate more buffers, increase the SLIH ring queue size, or to free up received buffers sooner.

Transmit Interrupts:
Total number of Transmit Complete interrupts.

Bytes Transmitted:
Total number of bytes transmitted.

Packets Transmitted:
Total number of packets transmitted.

Overflow Packets Received:
Total number of packets received whose length exceeded the size of the receive buffers.

VRM LLCs use the Send Command queue element to issue the **query statistics** command. The input queue element is defined as follows:

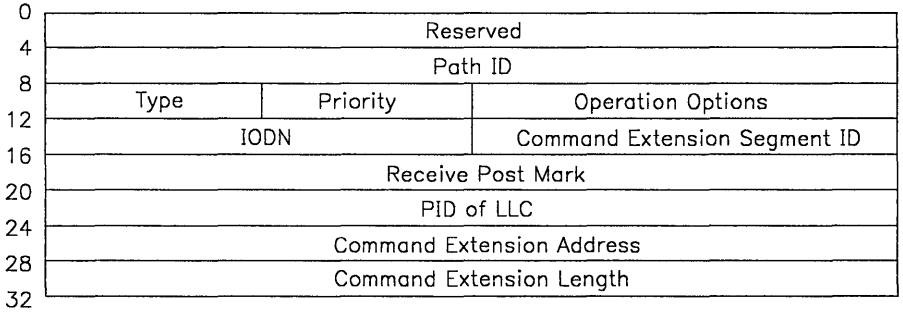


Figure 4-50. Query Statistics Queue Element

- Path ID - ID of the LLC-to-device driver path.
- Type - set to 1 for Send Command queue element.
- Priority - set to 0.
- Operation options - The command extension bit of the operation options field must be 1. The device option field is set to 9.
  - Bit 16 = Interrupt on completion (optional)
  - Bit 17 = Interrupt on error (optional)
  - Bit 18 = Synchronous operation (optional)
  - Bit 19 = Command extension (always 1)
  - Bit 20 = Reserved
  - Bit 21 = clear counters after **query statistics**
  - Bits 22-23 = Reserved

- Bits 24-31 = 9 for query statistics.

- IODN - of the adapter.
- Command Extension Segment ID - segment ID of the command extension area.
- Receive Post Mask - the ECB bit used when the device driver does a **_post** to the LLC to indicate received data.
- PID of LLC - the process ID of the LLC.
- Command Extension Address - the address of the command extension area.
- Command Extension Length - the length of the command extension

The command extension passes additional parameters (see Figure 4-36 on page 4-53). For VRM LLCs, the VM Interrupt Level/Sublevel field is set to 0.

Returned information for **_enqueue** is described in *VRM Programming Reference*. Additional returned information is contained in the returned command extension (see Figure 4-49 on page 4-97) and the acknowledgment queue element.

The acknowledgment queue element is defined as follows:

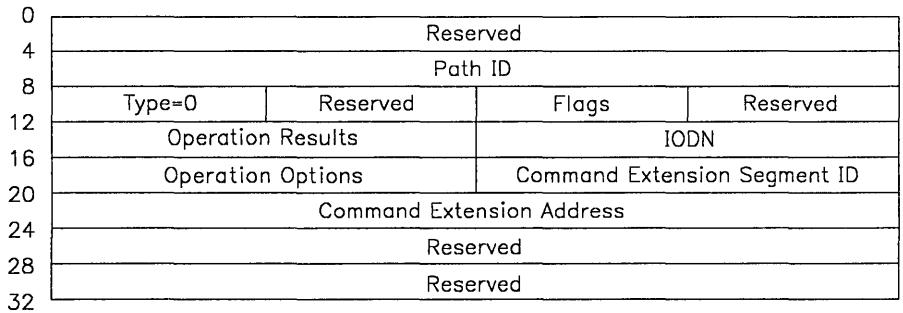


Figure 4-51. Acknowledgment Queue Element for the Query Statistics Command

Path ID: ID of the LLC-to-device driver path.

Type: Set to 0 for acknowledgment queue element.

Flags: 0x14 to indicate a solicited acknowledgment to a Send Command

Operation results:
See "Operation Results" on page 4-42.

IODN: Of the adapter.

Operation Options:
The operation options field from the Send Command queue element.

Command Extension Segment ID:
Segment ID of the command extension area.

Command Extension Address:
Address of the command extension area.

Receive Data

When the destination address of a packet matches the address of an adapter, the adapter receives that packet and places it in the adapter receive buffer. From the adapter receive buffer, the packet is moved (using DMA) to a buffer supplied by the token ring device driver. The data offset field of the driver's DDS device characteristics section indicates at what location in the device driver receive buffer to place the packet. This offset can be used to align the data portion of the packet on a fullword boundary to facilitate data transfers.

When a packet is placed in the receive buffer, the device driver checks the NetID in the type field for a match in the network ID correlation table. If the device driver does not find a match, the packet is discarded. If a match is found, the buffer address is placed in the device ring queue.

If the device ring queue was empty, the LLC is informed that a buffer was received. For VRM LLCs, a **_post** routine is used. For OS LLCs, a virtual interrupt is used. If the ring queue was not empty, no action is taken as the LLC is expected to empty the ring queue once it is informed that buffers are available.

Operation Results

Operation results for a command or unsolicited interrupt are defined in "Operation Results" on page 4-42.

The acknowledgment by the device driver to an LLC for the **start device**, **halt device**, and **write** commands are always solicited. These operation results are defined as follows:

0x0000: Successful operation
0x8200: Timeout error on transmit
0x8300: Internal command reject (device driver code problem)
0x8601: NetID table full
0x8602: Duplicate NetID in table
0x8801: Invalid queue element type
0x8802: Invalid command
0x8803: Invalid parameter
0x8804: Port is not open
0x8805: Port is already open
0x8806: Packet transmit error; retry the command
0x8C00: Open sequence failure; adapter failure
0x8D00: Unpin buffer problem.

If there is receive data for an LLC or an error occurs that is not related to the specified command, the device driver may send an unsolicited acknowledgment queue element or interrupt to the LLC. Operation results that the device driver returns for these conditions are as follows:

0x8100: Data available

0x8400: SLIH ring queue empty.

This indicates that a reconfiguration may be necessary to allocate more buffers, increase the SLIH ring queue size, or to free received buffers sooner.

0x8500: Device ring queue full.

This indicates that the LLC process is not keeping up with the receive data. A packet of data is lost for each occurrence of this result.

The device driver always sends solicited acknowledgments to the block I/O device manager for an **open** command. These operation results are defined as follows:

0x0000: Successful return code

0x8805: Port is already open.

Unsolicited queue elements are never sent to the device manager by the device driver.

Chapter 5. IBM Predefined Device Drivers

Contents

About This Chapter

This chapter describes the device drivers available for many types of VRM hardware support. These device drivers include:

- Asynchronous device driver
- Diskette device driver
- Fixed disk device driver
- Graphics asynchronous device driver
- Parallel device driver
- Small computer systems interface device driver
- Streaming tape device driver.

The define device structure of each device is defined, as well as the command interface and possible return codes.

Asynchronous Device Driver

The asynchronous device driver supports several hardware adapter cards and asynchronous devices. You can attach any supported device to any port on the adapter. This device driver supports the following IBM adapter cards:

- The serial port of the IBM Personal Computer AT® Serial/Parallel Adapter
- IBM RT PC 4-Port Asynchronous RS-232C Adapter
- IBM RT PC 4-Port Asynchronous RS-422A Adapter
- IBM RT PC Native RS-232 ports
- IBM RT PC 5080 Peripheral Adapter.

With the exception of the 5080 adapter, the asynchronous device driver allows connection of any supported device to any supported adapter. This driver supports only plotters using full duplex and XON/XOFF protocol from the 5080 adapter. For modems, however, switched and non-switched connection lines are supported from the other adapters.

All of the adapters support serial printers. The asynchronous device driver supports serial printers that employ XON/XOFF protocol on either the RS-232 or RS-422 adapters. The driver also supports RS-232-attached serial printers that use the direct connect protocol. For direct connect protocol, the device driver monitors its CTS line as a busy indicator.

In addition, all but the 4-Port Asynchronous RS-422A Adapter support the following IBM and Other Equipment Manufacturer (OEM) device types:

1. Modems
   - Full or half duplex
   - Switched or non-switched lines
   - Manual or automatic answering.
2. Terminals
   - ASCII
   - Direct or Remote.
3. Plotters
   - Full duplex
   - XON/XOFF protocol.

The 4-Port Asynchronous RS-422A Adapter supports terminals with direct-attach, RS-422A interfaces only.

Because the asynchronous device driver is device-independent, it provides no event log or problem determination support. Software above the VMI must provide event logging and problem determination support.

Operation Modes

The asynchronous device driver can run in either of two operation modes (mode 1 or mode 2). The operation mode you select determines how the driver interfaces to the virtual machine.

Mode 1 utilizes multiple types of unsolicited interrupts to inform the virtual machine of activity occurring on an adapter port. Mode 1 strips XON and XOFF characters from the received data stream. Mode 1 stores only received data bytes in the receive buffer.

Mode 2 uses the receive buffer to inform the virtual machine of much of the activity that occurs on an adapter port. The driver does not strip or substitute any received data. Mode 2 stores a 2-byte entry into the receive buffer for each received character, line status change, XON or XOFF received.

Other differences between the two operation modes can be seen in how each handles certain events. These events include:

- Received data (not XON or XOFF)

  Mode 1 stores a single data byte in the receive buffer.

  Mode 2 stores the received data byte and a status byte in the receive data buffer.

  The device driver generates an unsolicited interrupt any time the buffer goes from empty to not empty.

- Transmit data or DMA completion

  Both modes operate the same way. If more data remains to be sent, the device driver outputs the next data byte. If no more data remains to be sent, the driver generates a solicited interrupt to tell the virtual machine that the write command was completed.

- Received XOFF

  Mode 1 discards the XOFF and suspends transmission until it receives an XON.

  Mode 2 stores the XOFF and a status byte indicating that an XOFF was received in the receive data buffer. The driver also suspends transmission until it receives an XON. If the protocol is Direct Connect, the value stored in the data byte is 0x00.

  The virtual machine can issue a Halt Transmission command to make the driver purge the currently active write queue element.

- Received XON

  Mode 1 discards the XON and resumes transmission.

  Mode 2 stores the XON and a status byte indicating XON was received in the data buffer. The driver also resumes transmission. If the protocol is Direct Connect, the value stored in the data byte is 0x00.

- Modem status

  Both mode 1 and mode 2 report all modem status changes by way of an unsolicited interrupt.

The asynchronous device driver does not alter the state of output modem signals during a session (with the exception of dropping DTR to automatically disconnect the line for DSR inactive on a switched line connection). In any case, the driver continues to send or receive data (if already sending or receiving) when modem status changes.

- Line status

  Mode 1 generates an unsolicited interrupt for all line status changes. The driver stores received data in the receive buffer. The driver continues to transmit data if it was transmitting data when the line status changed.

  Mode 2 generates no interrupts for line status changes. The driver stores the received character and a status byte in the receive buffer. The driver continues to transmit data if it was transmitting data when the line status changed.

For either mode 1 or mode 2, if the receiver has not been enabled by a Read command, unsolicited interrupts are generated only for a detected break.

If a break is received after an XOFF but before an XON is received, the driver considers the break an XON and resumes transmission.

- Receive buffer overflow

  Mode 1 generates an unsolicited interrupt when the buffer overflows. Until free space is available in the buffer, subsequent received data bytes are placed in the last byte of the buffer, writing over the contents of this byte rather than wrapping to the beginning of the buffer.

  Mode 2 also generates an unsolicited interrupt to indicate buffer overflow. Until free space is available in the buffer, subsequent received data bytes are placed in the last two bytes of the buffer.

Regardless of mode, the device driver continues to send and receive data.

The Receive Buffer

You enable the receiver of the asynchronous adapter when you issue a **Start I/O SVC** with the Read option. Any data received prior to this time is discarded. The command element that follows specifies the starting address and length of a receive buffer. The length parameter, which must be at least 128 bytes, contains a pair of 32-bit pointers at the end of the buffer. The buffer must be aligned on a full word boundary at both its beginning and end points. The VRM pins this buffer in memory when the **Start I/O SVC** executes. To prevent the buffer from becoming unpinned during a session, the asynchronous device driver dequeues the Read command with the suppress option. The driver itself unpins the buffer at the end of a session when the driver processes the detach command (**Detach Device SVC**).

The last two words of the buffer contain a pair of in and out pointers. The first word is the input offset from the start of the receive buffer in which to place the next character. The driver maintains

a count for this word that increases by one each time a character is placed into the receive buffer. This pointer is wrapped as necessary by the device driver when the end of the buffer is reached.

The second word is the output offset from the start of the receive buffer. The virtual machine must maintain this word each time it moves or processes a character of information from the buffer and must also wrap the pointer if necessary when the end of the buffer is reached. Note that the end of the buffer is actually the buffer length minus the eight bytes that contain the pointer pairs. The virtual machine must also check for an empty buffer after increasing the output pointer and should not attempt to remove data if it detects an empty buffer.

In XON/XOFF protocol, the device driver monitors the pointers to determine when to send XON and XOFF. If the input pointer is greater than the output pointer, XOFF is sent if:

$$(In\_pointer - Out\_pointer) \geq Buffer\_size - XOFF\_threshold$$

If the output pointer is greater than the input pointer, XOFF will be sent when:

$$In\_pointer - Out\_pointer \leq XOFF\_threshold$$

In the examples above, buffer size is defined as the length parameter from the Read command element minus the eight bytes required for the pointers. XON is sent whenever the buffer becomes empty after an XOFF is sent.

Regardless of the protocol in use, the device driver checks for buffer overflow conditions and issues a virtual interrupt if the condition is detected. The driver detects an overflow condition when the input pointer equals the output pointer after the received character is stored and the input pointer is increased. The virtual machine should be aware that data loss is imminent when the driver detects an overflow condition.

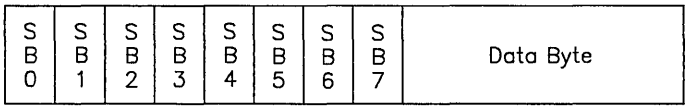
The virtual machine must also protect against missing an interrupt. You can do this by updating the output offset pointer when you receive data and checking for output offset equal to input offset when all data is received.

The device driver notifies the virtual machine by virtual interrupt when the two pointers are equal (buffer empty) prior to storing the new character in the receive buffer.

Selection of mode 1 or mode 2 has a direct effect on the virtual machine's use of the receive buffer. For mode 1, the device driver stores only received data bytes in the buffer. Unsolicited modem status, line errors, break detection, and received buffer overflow are reported to the virtual machine by way of virtual interrupts in mode 1.

For mode 2, the device driver stores a status byte for each data byte it receives. The virtual machine must typically set up a larger buffer for mode 2 operation, but the interface between the driver and the virtual machine is more efficient.

In mode 2, line errors, break detection, and received XON and XOFF signals are reported to the virtual machine through the receive buffer itself. The following figure shows the two bytes that will be stored into the receive buffer in mode 2 operation.



The fields in the preceding figure are defined as follows:

SB0 = Reserved and set equal to zero

SB1 = Reserved and set equal to zero

SB2 = Received XON

SB3 = Received XOFF

SB4 = Break detected

SB5 = Framing error

SB6 = Parity error

SB7 = Hardware-detected overrun.

When a good data byte is received, the driver sets no status bits. Only one status bit is ever set to indicate the interrupting condition. When multiple line errors occur, only the highest priority line error is reported. Error priority is assigned as follows:

1. Break detect (highest priority)
2. Hardware overrun
3. Framing error
4. Parity error (lowest priority).

When SB5, SB6, or SB7 are equal to one, the data byte contains the received data character that caused the line error.

When a break condition is detected on the receive data line, SB4 is set equal to one and the data byte is set to 0x00.

When the port is configured for a direct connect printer and the driver detects an XOFF sequence, SB3 is set equal to one and the data byte is set to 0x00. When the driver receives an XON sequence, SB2 is set to one and the data byte is set to 0x00.

Define Device Header

Because the asynchronous device driver supports such a variety of hardware configurations on several ports, the header fields are generally filled in at configuration or initialization time. For a definition of the format of the individual fields, see Figure 2-2 on page 2-5.

This device driver maintains an internal table to store the IODN and a pointer to the DDS of each port currently defined. The **_defind** routine places the IODN and DDS pointer in the table at an index based on the starting address of the port.

Because every asynchronous port is serially-reusable, only one DDS can be stored in the table for each port. To prevent a second DDS from writing over a DDS which is in use, the **_defind** routine rejects (with a 12 return code) any DDS for a port that already has a DDS defined. Only after you delete a DDS for a port will a new DDS be accepted for that port. Any subsequent **Attach Device SVC** or **Start I/O SVC** should be directed to the IODN of the new DDS.

Note: You must ensure that the adapters you are using with RT PC are configured correctly. In particular, no two adapter cards may be installed at the same I/O address range. For example, if you have two serial/parallel cards, one must be configured as a primary port, and the other one must be configured as a secondary port. For the multiport adapter, no two cards (whether two RS-232 cards, two RS-422 cards, or one of each) can be configured as the same card number. Valid card numbers are 1, 2, 3, and 4. If you do not follow these configuration restrictions, a hung system or system abend may result.

Hardware Characteristics

The asynchronous device driver can control three different adapter cards. Each adapter has slightly different hardware characteristics. You may want to refer to Figure 2-3 on page 2-6 for a look at the structure of this field.

Serial Port of the Serial/Parallel Adapter

Hardware characteristics for this adapter follow.

Length: 0x0000000F

Internal device type:

- I/O bus device
- Switchable to coprocessor
- 8-bit device
- 2 interrupt level definitions
- Adapter type = 0x23.

Base I/O port address:

- Primary serial port = 0x000003F8
- Secondary serial port = 0x000002F8.

Number of I/O port addresses: 8

Bus memory start/end address: N/A

DMA type: Not supported

Interrupt definition:

- Interrupts enabled
- Interrupts not shareable
- Primary port interrupt level = 4
- Secondary port interrupt level = 3.

Only one serial adapter can actively use a particular interrupt level at any given time. You can have a maximum of 2 serial adapters in use at a time.

Multiport Adapters (RS-232C, RS-422A)

Hardware characteristics for this adapter follow.

Length: 0x0000000F

Internal device type:

- I/O bus device
- Switchable to coprocessor
- 8-bit device
- 2 interrupt definitions
- Adapter type:
  0x35 (RS-232C)
  0x39 (RS-422A).

Base I/O port address:

You can use up to 4 of this type adapter card. The following list defines the possible addresses, where 'n' represents the card number (1-4).

- 0x0000n230 (port A of card n)
- 0x0000n238 (port B of card n)
- 0x0000n240 (port C of card n)
- 0x0000n248 (port D of card n).

Number of I/O port addresses: 8

Bus memory start/end address: N/A

DMA type: Not supported

Interrupt definition:

- Interrupts enabled and shareable
- Possible interrupt levels: 0x09, 0x0A, 0x0B.

You select the interrupt level on the adapter card itself. Because the adapter allows interrupt sharing, you can pick any of these levels for use with the multiport adapter cards.

Native RS-232C Adapters

Hardware characteristics for this adapter follow.

Length: 0x0000000F

Internal device type:

- I/O bus device
- Switchable to coprocessor
- 8-bit device
- 2 interrupt levels per adapter
- Adapter type = 0x10.

Base I/O port address:

- 0x00008001 (channel A - connector S2 on the machine)
- 0x00008000 (channel B - connector S1 on the machine).

Number of I/O port addresses: 0x0000001D

Bus memory start/end address: N/A

DMA type:

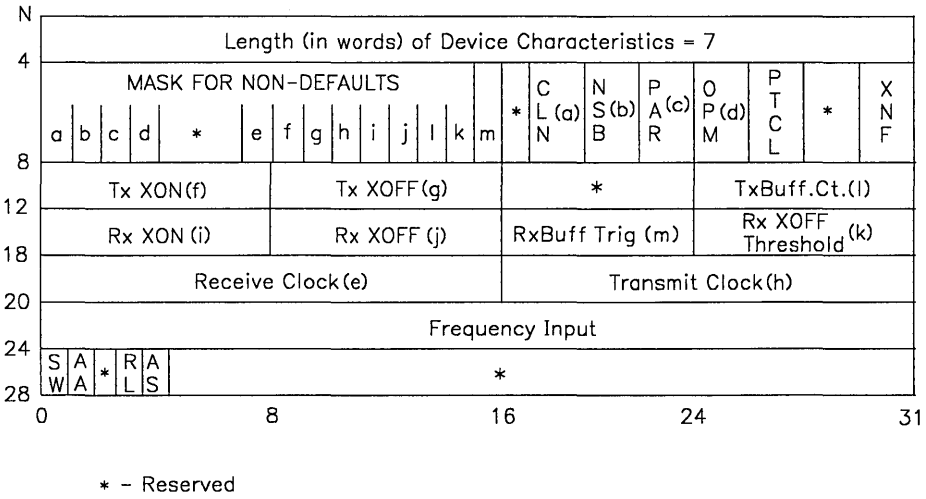
- DMA supported
- Channel enabled
- Number of windows = 0
- Channel numbers:
  0 = Channel A (Primary channel)
  1 = Channel B (Primary channel)
  2 = Channel A (Secondary channel)
  3 = Channel B (Secondary channel).

Interrupt definition:

- Interrupts enabled and shareable
- Interrupt levels:
  2 (interrupt level for the chip itself)
  13 (external logic interrupt level for these ports).

Device Characteristics

The asynchronous device driver device characteristics fields are shown in Figure 5-1.



* - Reserved

Figure 5-1. Asynchronous Device Characteristics

The parameters shown in the preceding figure apply to all asynchronous devices and applications except where noted. The device characteristics fields are defined as follows:

- Length

  The length of the device characteristics field for this driver is 7 words.

- Mask for non-defaults

  These bits serve as flags for other device characteristics values you can specify. A 0 in any of the fields a through m indicates that you want to use the default value for the corresponding field. A 1 in any of these bits means that you will supply a valid value for the corresponding field. For example, if you place a 1 in bit 0 ('a' in the preceding figure) of the mask area, you must supply a value for the corresponding field. In this case, the corresponding field is the character length field, bits 17-18 of the second word.

- CLN (character length)

  This 2-bit field indicates character length. Possible values are:

  00 = 5 bits per character

  01 = 6 bits per character

  10 = 7 bits per character (default)

11 = 8 bits per character

In the preceding figure, 'a' is the mask bit for this field. If 'a' is 0, the 7 bits-per-character default is used. If a is 1, select from the values defined above for this field.

The device driver expects to receive transmitted data right-justified in the byte with non-significant, high-order bits set equal to zero. Received data characters are also right-justified with non-significant high-order bits set equal to zero by the device driver. All data is sent or received with the least-significant bits first.

- NSB (number of stop bits)

  This 2-bit field defines the number of stop bits. Possible values are:

  00 = Reserved

  01 = 1 stop bit (default)

  10 = 1.5 stop bits

  11 = 2 stop bits

  'b' represents this field's mask bit in the preceding figure.

- PAR (parity checking and generation)

  This 3-bit field defines the parity checking (Rx) and generation (Tx) capabilities of this driver. Possible values are:

  000 = No parity

  001 = Odd parity

  010 = Mark parity

  011 = Even parity (default)

  100 = Space parity

  101 = Reserved

  110 = Reserved

  111 = Reserved.

  The native RS-232C ports cannot use mark or space parity. 'c' represents the bit mask for parity checking in the preceding figure.

- OPM (operation mode)

  This field defines the valid operation modes. Printer or plotter attachments must always use full duplex operation mode. Possible operation mode values are:

  00 = Reserved

  01 = Reserved

10 = Full duplex (default)

11 = Half duplex.

In the preceding figure, 'd' represents the bit mask for this parameter.

- PTCL (protocol)

  This field defines the requested protocol. In the following discussion of possible values for this parameter, the 'x' bit designates operation mode 1 or 2. (See "Operation Modes" on page 5-7.) A zero in the 'x' bit indicates mode 1. A one in the 'x' bit indicates mode 2. Possible values are:

  x00 = DTR modem control

  x01 = CDSTL modem control

  x10 = Direct connect printer protocol

  x11 = Reserved.

  This field has no default (or default bit mask).

- XNF

  This bit indicates whether XON/XOFF protocol should be included with the selected protocol. If you select half duplex mode, this bit must be 0.

- Tx XON

  This byte is compared to the received data if the XNF flag equals 1 and the transmit function has been disabled with an XOFF byte. If the comparands are equal, the driver resumes sending data. When this field equals 0xFF, the driver interprets any character following an XOFF as an XON.

  The default value for this field is 0x11, and the default bit mask in Figure 5-1 on page 5-14 is 'f'.

- Tx XOFF

  This byte is compared to the received data if the XNF flag equals 1 and the Tx function is enabled. If the comparands are equal, the driver stops sending data until it receives an XON character. The default value for this field is 0x13; the default bit mask in Figure 5-1 on page 5-14 is 'g'.

- Tx Buff. Ct.

  Certain asynchronous ports have the capability to buffer transmit data. Depending on the receive device, however, some data may be lost when the RT PC receives a pacing signal and all the buffered data is transmitted. This field allows you to specify that the device driver buffers fewer than the maximum number of data bytes supported by the adapter. Possible values for this field are 0x00 to 0xFF, and the default value is 0x10 (16 bytes buffered). The values 0x01 to 0x0F indicate 1 to 15 bytes buffered, while 0x00 and 0x10 and above indicate 16 bytes.

  For adapters that do not allow transmit-data buffering, this field is ignored.

  The default bit mask for this field is 'l'.

- Rx XON

  This field contains the value to transmit to inform a remote device that it can resume sending data.

  The default value for this field is 0x11, and the default bit mask in the preceding figure is 'i'.

- Rx XOFF

  This field contains the value to transmit to inform a remote device to stop sending data until it receives the XON character. The default for this field is 0x13; the default bit mask is 'j'.

- Rx Buff Trig

  For asynchronous ports whose adapter provides receive data buffering, this field indicates the number of bytes that will trigger a received data interrupt. When attaching devices that typically send a small amount of data to the host (such as printers and plotters), IBM recommends that a low trigger value, such as 1, be selected. When attaching to devices that typically send a large amount of data to the host (such as full-duplex communication lines), IBM recommends that you select a higher trigger value, such as 14. This field is ignored for adapters that do not provide a receive data buffering. Valid values for this field are:

Rx Buff Trigger	**Trigger Level (bytes)**
0x01	1
0x04	4
0x08	8
0x0E	14

  All other values are reserved. The default value for this field is 0x0E, and the default bit mask for this field is 'm'.

- Rx XOFF Threshold

  The value in this field indicates when the receive buffer is full (if the XNF flag equals 1). This value tells you how many free bytes are in the buffer when the driver sends an XOFF to the attached device. The receive XON threshold is fixed at 0 bytes.

  The default value for this field is 20 bytes, and the bit mask is 'k'. You can specify a value for this 8-bit field in the range 0 through 255.

- Receive clock

  This halfword contains the receive data baud rate for the adapter. Because the receive and transmit rates for supported adapters must be the same, this value must be set equal to the value of the transmit clock.

  The default value for this field is 9600 bits per second (bps), and 'e' represents the default bit mask in the preceding figure. See Figure 5-2 on page 5-20 for valid data rates.

- Transmit clock

  This halfword contains the transmit data baud rate for the adapter. The value in this field must match the value set for the receive clock.

  The default value for this field is 9600 bps, and the default bit mask is represented by 'h' in Figure 5-1 on page 5-14. See Figure 5-2 on page 5-20 for valid data rates.

- Frequency input

  This field contains timer input to the adapter interface. Two values are possible, depending on the adapter in use. For serial port of the IBM Personal Computer AT® Serial/Parallel Adapter, RS-232C and RS-422A adapters, the frequency input value is 1.8432 megahertz (Mhz.). For the native RS-232C ports, the frequency is 3.072 Mhz.

  You cannot modify these values with the change characteristics option of the **Start I/O SVC**.

- SW (switched or nonswitched lines)

  This field indicates whether the modem attachment uses switched or nonswitched lines. A 1 in this bit means switched lines, and a 0 means nonswitched lines.

  You cannot modify this field with a **Start I/O SVC**.

- AA (automatic answering)

  This field indicates whether automatic or manual answering mode is in effect for communications. A 1 in this field means automatic answering, and a 0 means manual answering.

  You cannot modify this field with the change characteristics option of the **Start I/O SVC**.

- RL (remote or local device attachment)

  This field indicates whether the attached device (plotter, printer, display, and so on) is attached locally (cabled directly to the host) or remotely (through a modem to the host). A one in this bit indicates remote attachment and a zero indicates direct attachment. Remote devices affect the modem status information in the PSB queue element. To meet an international requirement regulating data transmissions, all of CD, DSR, and CTS must be active in order for DSR and CD to be reported active. The CTS and RI status bits always reflect actual status, regardless of the RL setting. This attribute also helps the software above the VMI log errors. This field has no default and cannot be modified with the change characteristics option of the **Start I/O SVC**.

- AS (assert DTR and RTS signals to device)

  This field determines whether the device driver deactivates the DTR and RTS output control lines at the end of sessions to a locally attached device. This option is provided so that the lines may be left continually on (once they are activated by an **Open** command) for devices that require this.

  When this bit equals one, the lines are left active on **Close** and **Detach** commands. When this bit equals zero, the lines are dropped after **Close** and **Detach** commands. If the device has a remote connection (RL bit = 1), the lines are dropped regardless of the AS bit value. This field has no default and cannot be modified with the change characteristics option of the **Start I/O SVC**.

Valid Clock Values	Data Rates
0x0000	50
0x0001	75
0x0002	110
0x0003	134.5
0x0004	150
0x0005	300
0x0006	600
0x0007	1200
0x0008	1800
0x0009	2000
0x000A	2400
0x000B	3600
0x000C	4800
0x000D	7200
0x000E	9600
0x000F	19200

Figure 5-2. Asynchronous Data Rates

Error Log

Because of the number and variety of devices that can be used with the asynchronous device driver, this driver does not determine error severity or keep an error counter. The device driver does report changes in modem control status and line status to the virtual machine. The virtual machine is responsible for error determination and recovery, in any case.

CCB Input Values

The asynchronous device driver responds to a **Start I/O SVC** to schedule I/O. This SVC sends as a parameter the address of the command control block (CCB) for the work request.

The CCB contains information about the I/O operation. See *VRM Programming Reference* for a description of the CCB.

An important field in the CCB is the options field. This field is divided into operation and device options and indicates the type of I/O operation you want.

When the device driver receives a CCB to perform work, the driver's check parameters subroutine validates certain CCB fields. Depending on the device option you requested, the check parameters routine may find an error and return an error code.

Any of the options can return the following return codes:

0 = Operation successful

256 = Device option rejected because it is unrecognized.

257 = Device option rejected because the IODN is not valid.

Return codes that are specific to a requested device option are defined along with that device option.

Device options for the asynchronous device driver include:

0 = Read

The virtual machine must issue this command before it expects the device driver to accept any received data. Exactly one command element must follow. If you fail to attach a command element with this request, the check parameters routine will reject the request (return code = 261). You can issue this command only once per session (a session is the time from attach to detach). If you try to issue this command more than once during a session, the check parameters routine will reject it (return code = 262).

The Read command establishes a buffer in which to place received data bytes. See "The Receive Buffer" on page 5-8 for information on the interface to this buffer.

1 = Write

One or more command elements must follow a write command to indicate the address and length of each transmit buffer. If you fail to attach a command element with this request, the check parameters routine will reject the request (return code = 261). The native RS-232C ports have a transmit buffer size limit of 62K bytes. The buffer size for the other asynchronous adapters is unlimited. The driver outputs the data when it receives the write command and will transmit all chained data unless it receives a Halt Transmission command. A legal XOFF sequence suspends data transmission, and a legal XON sequence resumes it.

2 = reserved

3 = reserved

4 = Change device characteristics

No command elements follow this option. As described in "Device Characteristics" on page 5-14, you can change some of the default values found in the device characteristics section of the DDS. If you decide to change any defaults, bytes 8-23 (device-dependent parameters) of the CCB will have the same values as bytes 4-19 of the device characteristics.

Two restrictions govern the use of the change option. First, you cannot change the operation mode field (OPM) during a device session. A device session is the time between the issuance of the **Open** and **Close** commands. You must close a device session if you want to change the mode. The new mode goes into effect when you open the next device session. The other restriction involves the mode bit of the PTCLF field. This bit should not be changed during a device session. If this bit is changed during a device session, a system abend may result. You can change all of the other fields during a session.

You may receive the following return codes from the driver's check parameters routine when you request the change characteristics option:

280 = You chose a reserved (and therefore invalid) parameter for the number of stop bits field.

281 = You chose a reserved (and therefore invalid) parameter for the parity checking and generation field.

282 = You chose a reserved or otherwise invalid parameter for the protocol field.

283 = You chose a reserved (and therefore invalid) parameter for the receive clock field, or the receive clock value is not equal to the transmit clock value.

284 = You chose a reserved (and therefore invalid) parameter for the transmit clock field.

285 = You chose a reserved (and therefore invalid) parameter for the operation mode field.

286 = You chose an invalid combination of stop bits and bits per character. For example, 2 stop bits is invalid when you select 5 bits per character, and 1.5 stop bits is invalid when you select 6, 7, or 8 bits per character. This error does not apply to the native RS-232 ports.

287 = You chose an invalid combination of XNF and operation mode values. You cannot have XON/XOFF protocol when you are in half duplex operation mode.

288 = You chose a reserved (and therefore invalid) parameter for the Rx Buff Trig field.

289 = You chose an invalid combination of adapter port and parity checking and generation. You cannot use mark or space parity with the native RS-232C ports.

5 = Open

No command elements follow this option. In full duplex, the device driver opens the session by raising both data terminal ready (DTR) and request to send (RTS). The driver then generates a solicited virtual interrupt to report current modem signal status. Subsequent modem status changes generate an unsolicited interrupt.

A session is considered successfully opened when data set ready (DSR) and clear to send (CTS) respond to DTR and RTS. The driver can then accept other commands for the session. The virtual machine must set a timer to detect no modem status changes in response to DTR and RTS.

For half duplex, the driver actions are the same as for full duplex, except that only DTR is raised and only DSR indicates a successful open.

You can issue an open command only once per device session. If you attempt to issue an open command more than once during a session, the check parameters routine will reject it (return code = 258).

6 = Close

No command elements follow. For non-switched networks, the device driver closes the session by dropping DTR and RTS and generating a solicited virtual interrupt. The interrupt reports current modem signal status. Next, the driver disables adapter interrupts so that no further virtual interrupts will be generated.

For switched networks, the device driver closes a session by dropping DTR and RTS and setting a 10 second failsafe timer. If the DSR signal is detected as going inactive before the timeout occurs, the session is considered closed. A solicited virtual interrupt (with current modem signal status) is generated, then all adapter interrupts are disabled. If DSR is not detected prior to the timeout, a solicited virtual interrupt with current modem signal status is generated.

7 = Sendbreak

No command elements follow. This command forces serial output to the spacing state for approximately 500 milliseconds. The sendbreak command executes off the queue. Therefore, the virtual machine must clear the queue before issuing sendbreak to ensure that the break is initiated quickly. The sendbreak command is dequeued at the end of the time period for sending the break. A solicited interrupt indicating modem status is then sent to the virtual machine.

8 = Halt Transmission

No command elements follow. The virtual machine can use this command to halt transmission in the currently active write queue element. This command is used primarily to stop transmission when the device driver detects some condition that it cannot handle without the help of higher software levels. If the virtual machine

detects a condition that requires the cancellation of all pending operations, it should use a **Cancel I/O SVC** before it issues the Halt Transmission command. The SVC clears the driver's queue of all but the active queue element. Halt Transmission purges any data remaining from a write command, ends the write operation, and issues a solicited interrupt to report the current modem status. A Halt Transmission command has no effect if received when no write command is active, so the driver does not generate a virtual interrupt.

If the device driver was waiting for an XON from the device, the Halt Transmission resets the device driver with regard to the XON. Thus, the driver begins to transmit immediately when it receives the next write command.

You may receive the following return code with this command:

312 = Halt transmission command successful.

9 = Resync

No command elements follow. This command is processed exactly like the Close command except that the modem control lines to the device are not affected (not lowered as with the Close command) and the input control lines are not monitored to determine successful closing.

10 = Resume

No command elements follow. If the device driver is waiting for an XON from the device, the Resume command resets it with regard to the XON. Thus, the driver will begin transmission immediately if a Write command is pending.

You may receive the following return code with this command:

330 = Resume command successful.

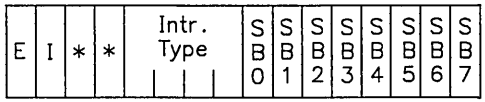
Output Values

The asynchronous device driver maintains certain status and operations results fields that it can copy into the Start I/O PSB. The virtual machine then has access to this information from the PSB.

The PSB fields that contain I/O completion information or detected error information are the status flags and the operation results fields. Figure 2-6 on page 2-12 shows the location of these fields in the PSB.

The status flags fields tells you right away with a bit mask if the I/O operation completed successfully. Bit 5 of the status flags field defines whether the interrupt is solicited or unsolicited. A solicited interrupt (bit 5 = 1) always means successful I/O completion. If the status flags field indicates unsuccessful I/O completion (bit 5 = 0), the virtual machine must examine the operations results field to determine the cause of the unsolicited interrupt.

The 16-bit operation results field provides error or informational data. Figure 5-3 on page 5-25 shows this field.



* Reserved (Set equal to 0)

Figure 5-3. PSB Operation Results Field

The bits in the operation results field are defined as follows:

- E — Error

  When this bit equals 1, the device driver is reporting an error. When this bit equals 0, no error occurred.
- I — Information

  When this bit equals 1, the device driver is relaying information to the virtual machine. One or the other, but not both, of bits 0 and 1 will be set equal to one when the driver reports operation results.
- Bits 2 and 3 are reserved.
- Intr Type — Interrupt Type

  This 4-bit field, along with the status bits, helps pinpoint the error-causing condition.
- Status Bits (SB0 through SB7)

  Status bits (SB) define a set of interrupt-causing conditions for each 4-bit interrupt type.

The following list defines the binary values of the 4-bit interrupt types and the status bit settings for each. Interrupt types 0011, 0100, and 0111 through 1111 are reserved.

Interrupt Type **Description**

0000 Receive interrupt without error

This informational message ('E' bit equals zero, 'I' bit equals one) informs the virtual machine that a byte has been placed in an empty receive buffer. All status bits equal zero.

0001 Receiver line status change

This informational interrupt occurs for devices operating in Mode 1 when the device driver detects a status change in the line. This interrupt is issued in Mode 2 only when a Break is detected before the Read command is issued.

For buffered adapters operating in Mode 1, only the last detected line status error is reported. For Mode 2, all line status errors are detected and reported. The status bits are defined as follows:

- SB0 through SB3 — Reserved
- SB4 — Break interrupt

  When SB4 equals one, the received data input was held in the spacing state for longer than a full word of transmission time.
- SB5 —Framing error

  When SB5 equals one, the received character did not have a valid stop bit.
- SB6 — Parity error

  When SB6 equals one and parity checking is enabled by the **Define Device SVC**, the driver detected an invalid parity on the received byte.
- SB7 — Overrun error

  When SB7 equals one, a character in the hardware receive buffer was lost. This interrupt pertains only to hardware receiver operations when the receiver fails to read the buffer before another character arrives.

0010 Modem status change

This is an informational interrupt. The asynchronous device driver generates modem status interrupts when any of the input modem signals (except Ring Indicator) change state. A state change for Ring Indicator is defined only as the transition from on to off. Modem status changes that occur during debouncing will not be reported.

Status bits are defined as follows:

- SB0 — Carrier detect status
- SB1 — Ring indicate status
- SB2 — Data set ready status

- SB3 — Clear to send status
- SB4 through SB7 — Reserved.

0101 Initialize device failed

The following error interrupts ('E' bit equals one, 'I' bit equals zero) occur when you set invalid choices in the device characteristics field of the define device structure (DDS). The device driver's initialize device subroutine detects these errors.

The 8 status bits (also known as the status byte) are set in combinations to specify errors. The hexadecimal values of the status byte and the error each represents follow. Values 0x04, 0x0A, and 0x0C through 0xFF are reserved.

- 0x00 — The adapter you selected is not detected currently on the machine.
- 0x01 — You chose a reserved option for the number of stop bits field.
- 0x02 — You chose a reserved option for the parity checking field.
- 0x03 — You selected a reserved or invalid option for the protocol field.
- 0x05 — You selected an invalid option for the receive clock field, or the receive clock value does not match the transmit clock value.
- 0x06 — You selected an invalid option for the transmit clock field.
- 0x07 — You selected a reserved option for the operation mode field.
- 0x08 — You selected an invalid combination of values for the number of stop bits and character length fields for either a serial/parallel or multiport adapter.
- 0x09 — You selected an invalid combination of values for the XON/XOFF protocol and operation mode fields.
- 0x0A — You selected a reserved value for the Rx Buff Trig. field.
- 0x0B — You selected an invalid value for the parity checking field for the native RS-232 ports.

0110 Miscellaneous event notification

The device driver can also detect the following error interrupts. Status byte settings follow. Any status byte value not included in the following list is reserved.

- 0x04 — Receive buffer overflow

  The driver detects this error interrupt when no more data can be received in the input buffer.
- 0x08 — Start I/O rejected

  The driver detects this error interrupt when it receives a **Start I/O SVC** and the device driver is in an invalid state. In this case, a write command was issued prior to an open command (between device sessions), or a sendbreak command was issued before the device was successfully opened.
- 0x0E — Fatal software error

  The driver detects this error when it receives an invalid return code from a system routine.

International Considerations

This section summarizes some of the restrictions, implementation characteristics, and dependencies of the asynchronous device driver.

- Connect Data Set to Line (CDSTL) and Data Terminal Ready (DTR)

  CDSTL and DTR are two options for operation of a single interchange circuit. In CDSTL operation, the circuit is not set active until either RI or DSR activates from the DCE. If RI activates without DSR, CDSTL is made active to await DSR, then dropped. If DSR activates, CDSTL is set active and the line is seized for the duration of the session.

  For DTR operation, the circuit is driven from the time of **Open**, indicating to the DCE that the DTE is ready to answer a call or transfer data.

  Note that the asynchronous device driver does not time the period that CDSTL waits for DSR. CDSTL can wait indefinitely for DSR. If a timer is required to await DSR active modem status, the virtual machine must implement it.

  The RT PC does not support CDSTL in Japan and Italy since those countries specify modems that provide a level RI signal and the RT PC requires a pulsed RI signal. DTR mode works properly with a level or a pulsed RI signal and is supported in both Japan and Italy.

- Data Set Ready (DSR) debounce

  For switched network operation, the asynchronous device driver provides a 50-millisecond timeout after DSR is first detected on. This allows the DSR circuit state to be checked after it settles down.

  Note that the driver deglitches the circuit only when the circuit initially comes active. Subsequent modem or DSR changes are then passed to the virtual machine.

- Prevention of false disconnect

  Typically, a 200-millisecond delay between a close and a subsequent open of a line is provided to ensure that the line is fully released. Note that the asynchronous device driver does not provide this delay. If such a delay is required, the virtual machine must implement it.

- Automatic disconnect

  For switched network operation, the capability to automatically disconnect a call should be provided in the following situations:

  - Drop of DSR for more than 50 milliseconds when a link is in session (data security violation)
  - Completion of a DISC sequence
  - Abnormal condition, such as excess line quiet period.

The asynchronous device driver provides automatic disconnect for a switched line if DSR drops. If DSR drops, the device driver also drops DTR (disconnecting the line), and reports the states of DSR and CD as inactive. Programs that monitor DSR or CD to ensure a good connection can then terminate the transmission of data. The device driver continues to send and receive data after a disconnection until directed to stop. The virtual machine must issue a **Close** command before attempting to re-open a session from which the driver automatically disconnected.

Manual disconnection is allowed through the AIX Operating System **pdisable** command.

Note that the device driver does not attempt to time intervals of line quiet. If this timing is required, it must be provided by any program that allows automatic answering by an unattended connection.

Diskette Device Driver

This device driver supports one or two diskette drives attached to the Fixed-Disk and Diskette Drive Adapter. The first diskette drive must be an IBM Personal Computer AT® High Capacity Diskette Drive. The second drive can be an IBM Personal Computer AT® High Capacity Diskette Drive or IBM Personal Computer AT® Dual-Sided Diskette Drive. Both diskette drives are driven by a single copy of the diskette device driver. Each diskette drive has its own IODN although they have the same hardware characteristics. This is a nonshared device driver, which means that each diskette drive can only be attached to one virtual machine at a time.

This device driver can read and write the following types of diskettes:

- Double sided diskettes when used with the Personal Computer AT Dual-Sided Diskette Drive
- High capacity diskettes when used with the Personal Computer AT High Capacity Diskette Drive.

This device driver can also read double sided diskettes using the Personal Computer AT High Capacity Diskette Drive if the data was written using a Personal Computer AT Dual-Sided Diskette Drive. Writing to a double sided diskette using a Personal Computer AT High Capacity Diskette Drive is unpredictable and is not recommended.

The diskette device driver DDS is created by an lpost that passes the DDS to the loadlist processor. For information on the loadlist processor, see "VRM IPL and Configuration" on page 1-4. The diskette device driver is defined on the following pages.

Define Device Header

This section of the DDS is common to all device drivers. The information is defined as follows:

IODN: 4 or 5

IOCN: 0x0240

Define option: 0x0101

Device type: 0x0101

Device name: This field is ignored by the VRM. It can be used to store a name for the device that uses this device driver.

Offset to hardware characteristics:
28 bytes

Offset to device characteristics:
72 bytes

Offset to error log:
100 bytes.

Hardware Characteristics

This section of the DDS is common to all device drivers. The information is defined as follows:

Length: 11 words

Internal device type:

- 0xD1xx5200 for the first drive
- 0xD1xx5201 for the second drive.

(where xx indicates the slot occupied by the adapter).

Base I/O port address:
0x000003F2

Number of I/O port addresses:
6

Bus memory start address:
Not used by this device driver.

Bus memory end address:
Not used by this device driver.

DMA type: 0x8A000002

Interrupt type:
0x20000106

- Interrupts not enabled
- Interrupts not shareable
- Interrupt level = 6.

Device Characteristics

The Personal Computer AT High Capacity Diskette Drive can read double sided or high capacity diskettes. The device driver automatically adjusts the device characteristics depending on the type of diskette inserted in the diskette drive. The device driver uses the following procedure to determine which device characteristics to use in performing a read or write operation:

1. When the machine is initialized, the loadable POST determines the type of diskette drives installed and sets the device characteristics to match the diskette types. In most cases, this will be the Personal Computer AT High Capacity Diskette Drive.
2. If the device driver cannot read or write to the diskette, it changes to the default device characteristics for the Personal Computer AT Dual-Sided Diskette Drive.
3. If the device driver still cannot read or write to the diskette, it moves the read/write head to track 0, resets the device characteristics to the original values, and returns a -11 in the operation results field of the PSB.
4. Once the device characteristics are changed, they remain in effect until a diskette with a different format is inserted in the drive or a change characteristics device option is issued to the device driver through a CCB.

This automatic switching can be turned off by setting Bit 7 (the eighth bit) in the options field of the CCB when attempting to read or write to the diskette. This tells the device driver not to reset the device characteristics upon receiving a read or write error. Two common causes for a read or write error are inserting the wrong diskette type in the diskette drive and inserting an unformatted diskette in the diskette drive.

Note: As long as you are performing operations on standard double-sided or high-capacity diskettes, you can ignore the automatic switching function as it is performed by the device driver. However, if you perform any operation that uses non-standard device characteristics, you should turn off the automatic switching by setting Bit 7 in the options field of the CCB.

When bit 7 of the CCB operations option field is set and a read or write error is detected, the device driver returns a -11 and the virtual machine must decide how to handle the condition.

This procedure is the only checking done by this device driver with regard to device characteristics. Any changes other than the automatic switching between the two default diskette formats must be explicitly performed using a change characteristics device option. This is defined in "CCB Input Values" on page 5-40.

The device characteristics for this device driver are shown in Figure 5-4 on page 5-34.

0	Length			
4	Sector Size	Sectors/Trk	Trks/Cylinder	# of Cyls
8	Device Type	Step Rate	Head Load	Head Unload
12	Motor Start	Head Settle	R/W Gap	Format Gap
16	Fill Byte	VFO	Reserved	Reserved
20				

Figure 5-4. Device Characteristics

The default values for each characteristic are shown below. The system may change these values with the change characteristics device option. Note that the device driver does not check the changed characteristics for validity. It will attempt to use whichever characteristics are given to it. To re-establish the default values, the virtual machine can issue a change characteristics device option with the default values or perform a restart.

Length: 5 words

Sector size: Can be one of three values:

- 01 — 256 bytes
- 02 — 512 bytes
- 03 — 1024 bytes.

The diskette drives can use any of the three values listed. However, this device driver only supports 02. Therefore, all diskettes used by this device driver must use 512-byte sectors.

Sectors/Track:

- 8 or 9 — Personal Computer AT Dual-Sided Diskette Drive
- 15 — Personal Computer AT High Capacity Diskette Drive.

Trks/Cylinder: Tracks per Cylinder

- 1 — Personal Computer AT Dual-Sided Diskette Drive using single-sided diskettes
- 2 — Personal Computer AT Dual-Sided Diskette Drive using double-sided diskettes
- 2 — Personal Computer AT High Capacity Diskette Drive.

# of Cyls: Number of Cylinders

- 40 — Personal Computer AT Dual-Sided Diskette Drive
- 80 — Personal Computer AT High Capacity Diskette Drive.

Device Type:

- 1 — Personal Computer AT Dual-Sided Diskette Drive
- 2 — Personal Computer AT High Capacity Diskette Drive.

Step Rate:

- 12 — Sets the Personal Computer AT Dual-Sided Diskette Drive step rate to 6 milliseconds.
- 13 — Sets the Personal Computer AT High Capacity Diskette Drive step rate to 3 milliseconds.

Head Load Time:

- 1 — Sets the Personal Computer AT Dual-Sided Diskette Drive head load time to 2 milliseconds.
- 25 — Sets the Personal Computer AT High Capacity Diskette Drive head load time to 50 milliseconds.

Head Unload: 15 — Sets both diskette drive head unload times to 240 milliseconds.

Motor Start:

- 4 — Sets the Personal Computer AT Dual-Sided Diskette Drive motor start time to .5 second.
- 8 — Sets the Personal Computer AT High Capacity Diskette Drive motor start time to 1 second.

Head Settle: 25 — Sets both diskette drive head settle times to 25 milliseconds.

R/W Gap Length:

- 42 bytes — Personal Computer AT Dual-Sided Diskette Drive
- 27 bytes — Personal Computer AT High Capacity Diskette Drive.

Format Gap:

- 80 bytes — Personal Computer AT Dual-Sided Diskette Drive
- 84 bytes — Personal Computer AT High Capacity Diskette Drive.

Fill Byte: This value is written on all data bytes when a diskette is first formatted. The value is 0xF6 for both diskette drives.

VFO: This value specifies the frequency at which the adapter will perform a read or write operation. It can be set to one of three values:

- 0 — 500K bytes per second. This is used to read or write high capacity diskettes with a Personal Computer AT High Capacity Diskette Drive.
- 1 — 300K bytes per second. This is used to read double sided diskettes with a Personal Computer AT High Capacity Diskette Drive.
- 2 — 250K bytes per second. This is used to read or write double sided diskettes with a Personal Computer AT Dual-Sided Diskette Drive.

The default values are:

- 2 — Personal Computer AT Dual-Sided Diskette Drive
- 0 — Personal Computer AT High Capacity Diskette Drive.

Error Log

The format of the error log for this device driver is shown in Figure 5-5.

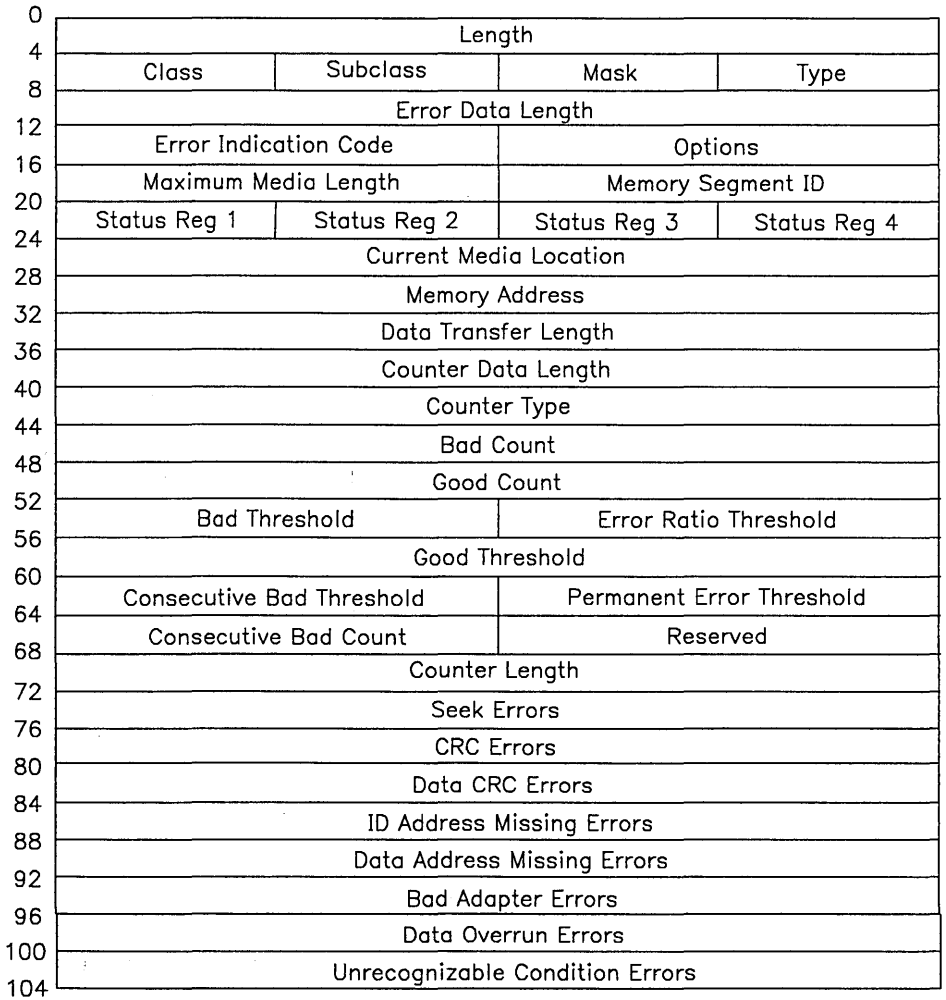


Figure 5-5. DDS Error Log Structure

This particular error log is logically divided into four sections, each of which is used for a different purpose:

- The first section is always used when an error occurs. It contains the standard error log information such as length, class, subclass, mask, and type.
- The second section starts at byte 8 and is only used when a temporary or permanent error occurs. This is indicated in the type field with a value of 0x40 or 0x80. This section contains information about the current I/O operation and the device that uses this device driver.
- The third section starts at byte 36 and is only used when an error causes a predefined error ratio threshold to be exceeded. This is indicated in the type field with a value of 0x20. This section contains information about the error that caused the ratio to be exceeded.
- The fourth section starts at byte 68 and is also only used when an error causes a predefined error ratio threshold to be exceeded. This section contains a list of counters that identify the total number of errors encountered for each type of error that can occur for this device driver.

"DDS Error Log" on page 2-6 describes how device drivers can be designed to monitor errors using this type of error log structure. You may need to read that section to understand the purpose of the fields of this error log. The various fields of the error log are defined as follows:

Length: 26 words

Class: 0x01 to indicate a hardware error, 0x02 to indicate a software error

Subclass: 0x03

Mask: 0x00

Type: Can be one of three values:

- 0x80 indicates that a permanent error occurred. This means that the second section will be used.
- 0x40 indicates that a temporary error occurred. This means that the second section will be used.
- 0x10 indicates that the error ratio threshold was exceeded. This means that the third and fourth sections will be used.

Error Data Length:
7 words

Error Indication Code:
Uses bits 7 through 15 to represent one of nine different types of errors that can occur. The bits are defined as follows:

- Bit 7 — Unrecognized condition
- Bit 8 — Data overrun error

- Bit 9 — Bad controller
- Bit 10 — Bad data address mark
- Bit 11 — Bad ID address mark
- Bit 12 — CRC error in data
- Bit 13 — CRC error
- Bit 14 — Timeout error
- Bit 15 — Seek error.

Options: The field contains the operation option and device option sent to the device driver.

Maximum Media Length:
The maximum number of sectors on the diskette. This can be computed using the following formula:

```
Max = (Sectors per Trk) * (Trks per Cylinder)
      * (# of Cylinders)
```

Memory Segment ID:
The segment register contents at the time of the error.

Status Registers 1 through 4:
The values returned depend on the type of error that has occurred. The errors that return information in the status registers are listed below along with the register contents for each error.

- Unrecognizable condition

  The status registers contain the contents of ST0, ST1, ST2, and ST3. The first three values may not reflect the current operation. ST3 contains the condition of the drive at the time of the time-out error.

- Data overrun error

  Status Registers 1, 2, and 3 contain the contents of ST0, ST1, and ST2. Status Register 4 is not defined.

- Bad controller

  Status Registers 1, 2, and 3 contain the contents of ST0, ST1, and ST2. Status Register 4 is not defined.

- CRC Error

  Status Registers 1, 2, and 3 contain the contents of ST0, ST1, and ST2. Status Register 4 is not defined.

- Seek error

  Status Register 1 contains the contents of ST0. Status Register 2 contains the device driver's best estimate of the cylinder location of the read/write head.

Current Media Address:
The location on the diskette where the error occurred. This is made up of three bytes as shown:

- Byte 0 — Cylinder number
- Byte 1 — Head number
- Byte 2 — Sector number.

Memory Address:
Location of the memory buffer in use at the time of the error.

Data Transfer Length:
The number of bytes that were to have been transferred either to or from a memory buffer.

Counter Data Length:
The number of words used for this section of error data. For this device driver, the value is 8 words.

Counter Type:
The first bit in this field is always set to one. This field also uses bits 23 through 31 to represent nine different types of errors that are monitored by this device driver. One of these bits is set to one to indicate which error caused the error ratio threshold to be exceeded. Bits 23 through 31 are defined as follows:

- Bit 23 — Unrecognizable condition
- Bit 24 — Data overrun error
- Bit 25 — Bad adapter card error
- Bit 26 — Bad data address mark error
- Bit 27 — Bad ID address mark error
- Bit 28 — CRC data error
- Bit 29 — CRC error
- Bit 30 — Not used
- Bit 31 — Seek error.

Bad Count: Total number of errors encountered for the type of error specified in the counter type field.

Good Count: Number of successful I/O operations performed. The error ratio threshold is not checked until this value reaches 40. This ensures that enough operations have been performed to obtain meaningful results from an error ratio check.

Bad Threshold:
: Number of retries that must occur before checking the error ratio threshold. The value for this device driver is 3.

Error Ratio Threshold:
: The value which will cause the error counter data to be reported. This value is obtained by dividing the number of errors for a particular type of error by the good count. This device driver sets the threshold at 5, meaning 5 percent.

Good Threshold:
: The maximum number of good operations which can be used to compute the error ratio. If the good count counter reaches this value, it is reset to 0 by the device driver. This ensures that the good count value does not become so large that the error ratio becomes statistically invalid. The value for this device driver is 800.

Consecutive Bad Threshold:
: Number of consecutive retries that must occur before an error is reported. The value for this device driver is 3.

Permanent Error Threshold:
: Number of retries that must occur before logging a permanent error. The value for this device driver is 10. If an I/O operation reaches this value, the operation is abandoned and reported as unsuccessful.

Consecutive Bad Count:
: Number of consecutive retries that occurred for the last I/O operation. This field is only used for temporary errors. The value for this device driver can range from 3 to 9.

Counter Length:
: Nine words. This section is used to report the totals for each type of error when the error ratio threshold is exceeded.

Note that all of the error counters are reported in the last 8 words, even though only one particular type of error caused the counter data to be reported.

CCB Input Values

This device driver accepts CCBs as shown in Figure 2-5 on page 2-12. The CCB consists of the command header and zero or more command elements, depending on the device option used.

This device driver uses the following definitions for the bits in the options field of the CCB.

- Bit 0 through 4 — Operation option

- Bit 5 — Reserved
- Bit 6 — Reserved
- Bit 7 — Disable automatic device characteristic switching
- Bit 8 — Not used
- Bit 9 — Reserved
- Bit 10 — Enable read verification for read operations
- Bit 11 through 15 — Device option.

Most of the device options use the word that starts at byte 8 of the CCB header to define a field called the logical block address field. This field can contain a value from zero to n, where zero is the first sector of the first cylinder and n is the last sector of the last cylinder. The value of n depends on the format of the diskette being operated upon. The formula used to determine the logical address of a sector is shown below:

Logical Address = (Track Number * Sectors per Track
* Tracks per Cylinder) + Sector Number

When the device driver receives a device option that uses a logical address, it translates the address into a sector ID. Each sector on the diskette is identified by a four-byte sector ID that is stored in the first four bytes of that sector. The bytes are defined as follows:

- Byte 1 — Cylinder number
- Byte 2 — Head number

  The head number is either 0 or 1, depending on whether you are working with the first or second track on the cylinder.
- Byte 3 — Current sector
- Byte 4 — Sector size. This device driver sets the sector size to 512 bytes per sector. The value corresponding to 512-byte sectors is 2.

Device option 4, which allows you to change device characteristics, uses CCB header bytes 8 through 21 to store values for the various device characteristics.

The device options are defined below. Device options that require a command element contain information about the command element in the definition. If there is no information about a command element, you can assume that the device option does not use a command element.

0 = Read

This device option reads information starting at the sector identified by the logical address field. This device option uses a CCB element with the following field definitions:

Data Transfer Length:
Any integral multiple of 512 up to the limit of the DMA.

Memory Address:
Location of the data buffer that will receive the sector data.

This device option uses information supplied by the device characteristics fields in the DDS to determine how to read the diskette. You may need to change the device characteristics if a different diskette type has been inserted into the drive, or if the characteristics have been altered by a previous device option.

The virtual machine can determine the diskette type by reading the diskette descriptor on the diskette, or by trying to read sectors 8, 9, and 15 of any track. The results of these read operations will identify the type of diskette.

1 = Write

This device option writes information starting at the sector identified by the logical address field. This device option uses a CCB element with the following field definitions:

Data Transfer Length:
Any integral multiple of 512 up to the limit of the DMA.

Memory Address:
Location of the data buffer that contains the data to be written to the diskette.

See the description of the Read device option for information about device characteristics and diskette types.

2 = Position

This device option positions the read/write head at the cylinder whose first sector is stored in the logical address field of the CCB header. To compute the logical address of the first sector of the cylinder, use the following formula:

Logical Address = Track Number * Sectors per Track * Tracks per Cylinder

Note that any sector on the track will position the read/write head properly. This system uses the first sector as a matter of convenience.

3 = Format Track

This device option is used by the virtual machine to format one track of a diskette. The track is identified by the logical address field in the CCB. This device option writes a four-byte sector ID at the beginning of each sector on the specified track. This sector ID is then used for all read/write operations. Since the sector IDs do not use data space, the sector still has 512 bytes available for data.

This device option also sets each data byte in each sector to the value specified by the fill byte device characteristic. This system uses 0xF6 as the fill byte value.

The sector IDs for the track sectors are stored in memory and pointed to by a single CCB element. There is one sector ID for each sector on the track. Each ID on a particular track is the same except for the current sector value, which is 1 for the first sector ID and increases by one for every additional sector on the track.

The CCB element has the following field definitions:

Data Transfer Length:
: Number of bytes that contain sector ID data. This number must be a multiple of four since each sector ID is four bytes long.

Memory Address:
: Location of the data buffer that contains the sector IDs.

The virtual machine uses values stored in the device characteristics fields of the DDS to create the sector ID buffer. The DDS fields may need to be changed to the proper device characteristics before the buffer is created.

4 = Change Device Characteristics

This device option modifies the default device characteristics stored in the device characteristics section of the DDS. If the system is initialized or restarted, the device characteristics are reset to their default values.

After using the change device characteristics command, Bit 7 (disable automatic device characteristic switching) should be set in the options field of the CCB associated with every subsequent command. Otherwise, the device driver may automatically change back to the default characteristics when attempting to recover from errors.

This device option uses bytes 8 through 21 offset from the start of the CCB header to change the device characteristics. Note that this option uses the logical address field plus 10 additional bytes to store the device characteristics. The byte assignments are listed below:

- Byte 8 — Sector size
- Byte 9 — Sectors/Track
- Byte 10 — Trks/Cylinder
- Byte 11 — # of cyls
- Byte 12 — Device type
- Byte 13 — Step rate
- Byte 14 — Head load time
- Byte 15 — Head unload
- Byte 16 — Motor start
- Byte 17 — Head settle
- Byte 18 — Read/Write gap length
- Byte 19 — Format gap
- Byte 20 — Fill byte
- Byte 21 — VFO.

With the exception of byte 21, a value of zero in a byte indicates that no change is requested. For byte 21, which defines the read and write rate for data transfers, a value of zero is a valid parameter. Therefore, byte 21 is always copied into the VFO field of the device characteristics section of the DDS.

5 = Reset Drive

This device option sets the read/write head to track zero. This device option also sets an adapter register called present cylinder number to zero.

6 = Read Status

This device option reads the status of the last device option issued to the diskette drive adapter. This command uses a CCB element to point to an area in memory for the status information to be read into. The data transfer length field of the CCB element should be set to 9 bytes. The bytes are defined as shown below:

- Byte 0 — Last device option issued to the adapter by the device driver.
- Byte 1 — Status register 0. Contains the completion status of the operation.
- Byte 2 — Status register 1. Contains the first byte of the operation result.
- Byte 3 — Status register 2. Contains the second byte of the operation result.
- Byte 4 — Cylinder location when finished.
- Byte 5 — Head location when finished.
- Byte 6 — Sector location when finished.
- Byte 7 — Set to 2 to indicate 512 bytes per sector.
- Byte 8 — Status register 3. Contains the status of the diskette drive. The bits are defined as follows:
  Bit 0 — Not used.
  Bit 1 — Diskette is write-protected.
  Bit 2 — Not used.
  Bit 3 — Drive is at track 0.
  Bit 4 — First of two bits used to indicate head being used. Always set to 0.
  Bit 5 — Second of two bits used to indicate head being used. 0 indicates head 0; 1 indicates head 1.
  Bit 6 — Head address.
  Bit 7 — Diskette is double-sided.

7 = Check Diskette Change Status

This device option shows whether or not a diskette has been removed and a new diskette placed in the drive. It also shows whether or not the original diskette was removed and replaced. For this device option, removed is defined as pulling the diskette out of the drive far enough to set a sensor switch in the drive. The drive will signal that the diskette was removed until the read/write head is moved.

If the diskette has been removed, this device option returns a value of -15 in the operation results field of the Start I/O PSB. If the drive does not contain a diskette, a time-out error will occur when the next I/O operation is performed.

8 through 31 = Not used by this device driver.

Note:

This device driver does not contain a check parameters routine. Any required parameter checking is performed by the I/O initiate routine.

After successfully completing any of the device options, the device driver issues a dequeue function call and waits for the next command. The dequeue function call tells the queue manager that it can release the current queue element.

If the device driver cannot successfully complete a device option, the device driver issues an enqueue function call, which generates an interrupt request to the virtual machine. The current queue element is retained for further processing.

Output Values

This section defines the output values that may appear in GPR2 or in a PSB. The PSB fields that contain I/O information or detected error information are the status flags field and the operation results field. Figure 2-6 on page 2-12 shows the location of these fields in the PSB.

The status flags field is not used for this device driver.

The 16-bit operation results field returns a value indicating the completion status of the operation. The possible errors that can be returned are listed below:

00 = Successful completion

-02 = Seek error

A seek error was detected.

-03 = Bad DMA

Unable to resolve a DMA setup condition.

-04 = Write protect

Diskette is write-protected.

-05 = Timeout

The adapter would not respond to commands within the allowed time interval.

-06 = Bad drive

An adapter error occurred that prevented the diskette controller from operating properly.

-07 = CRC error in data

A CRC error was detected in the data area.

-08 = CRC error not in data

A CRC error was detected in the memory address or sector ID.

-09 = ID not found

Unable to locate the sector ID within the allowed time interval. This could be due to one of two reasons:

- The sector ID had a CRC error.
- The sector ID was not on the diskette or was incorrect.

If this condition occurs, the position of the read/write head is checked. If the head is at the wrong position, it is repositioned and the operation is tried again.

-10 = Bad data address mark

Unable to locate the data address mark in the data area within the allowed time interval. If this condition occurs, the position of the read/write head is checked. If the head is at the wrong position, it is repositioned and the operation is tried again.

-11 = Bad ID address mark

Unable to locate an address mark in the sector ID area. This may be due to a mismatch between the device characteristics and the diskette type. It can also occur if an unformatted diskette is placed in the diskette drive.

-12 = Unrecognized condition

An error occurred that the diskette driver could not correct or evaluate.

-13 = Equipment check

A hardware error was detected.

-14 = Data overrun

An error occurred that caused the data to be transferred at the wrong rate.

-15 = Diskette changed

The diskette was removed and replaced by the same diskette or a different diskette.

-16 = Reserved

-17 = Threshold exceeded

An error threshold has been exceeded. The operation that receives this response may have completed successfully; however, the command should be performed again to ensure proper completion.

-256 = Invalid device option

The device option in the options field of the CCB was not valid for this device driver.

-257 = Undefined device option

The device option in the options field of the CCB was not defined for this device driver.

-258 = Invalid parameters

The CCB contained an invalid parameter such as an invalid logical address.

Fixed-Disk Device Drivers

The RT PC supports as many as three fixed disks per machine and several fixed-disk adapters. Supported adapters include the RT PC Personal Computer AT Fixed-Disk and Diskette Drive Adapter, the RT PC ESDI Magnetic Media Adapter, and the RT PC High-Function ESDI Magnetic Media Adapter. The Personal Computer AT Fixed-Disk and Diskette Drive Adapter and ESDI Magnetic Media Adapter can run two fixed-disk drives per adapter card, and the RT PC allows up to two of these cards per machine. The High-Function ESDI Magnetic Media Adapter can run three fixed-disk drives and two diskette drives.

The VRM provides device driver support for all these fixed-disk drive adapters. One VRM device driver supports both the Personal Computer AT Fixed-Disk and Diskette Drive Adapter and the ESDI Magnetic Media Adapter adapters, and another driver supports the High-Function ESDI Magnetic Media Adapter. The High-Function ESDI Magnetic Media Adapter requires a separate device driver because this adapter provides both a programmed I/O mode and a DMA mode. The other adapters operate only in programmed I/O mode.

At IPL time, the power-on self tests (POSTs) determine the adapters that are configured on the machine. A DDS is then built for the device driver code (and given an IOCN) and the device is defined to the system (and given an IODN). Note that both fixed-disk device drivers may be present in the same machine if the two types of adapters are configured. Requests to the various fixed-disk devices are then determined by the IOCN and IODN associated with the request.

If the High-Function ESDI Magnetic Media Adapter is configured with both fixed-disk and diskette devices, both a fixed-disk and diskette device driver are required.

A fixed-disk device driver is different from the other device drivers in that it can receive commands from more than one source. It can receive CCBs from a virtual machine and it can receive internal VRM queue elements from the minidisk manager, the Personal Computer AT Coprocessor option, and virtual memory manager. CCBs and internal VRM queue elements contain similar information; however, there are some differences. This is explained in greater detail in "CCB Input Values" on page 5-58.

Define Device Header

This section of the DDS is common to all device drivers. The information for the two device drivers is defined as follows:

IODN: 01, 02, or 03

IOCN: 0x0230 for the Personal Computer AT Fixed-Disk and Diskette Drive Adapter or ESDI Magnetic Media Adapter

0x0231 for the High-Function ESDI Magnetic Media Adapter

0x232 for the High-Function ESDI Magnetic Media Adapter in the second slot.

Define option: 0x0101

Device type: 0x00F3

Device name: This field is ignored by the VRM. It can be used to store a name for the device that uses this device driver.

Offset to hardware characteristics:
28 bytes

Offset to device characteristics:
72 bytes

Offset to error log:
132 bytes.

Hardware Characteristics

This section of the DDS is common to all device drivers. The information is defined as follows:

Length: 11 words

Internal device type:

Personal Computer AT Fixed-Disk and Diskette Drive Adapter:

- 0x91xx5200 for the first drive of the first adapter card
- 0x91xx5201 for the second drive on the first adapter card
- 0x91xx5202 for the first drive on the second adapter card.

ESDI Magnetic Media Adapter:

- 0x91xx8200 for the first drive
- 0x91xx8201 for the second drive
- 0x91xx8202 for the third drive.

High-Function ESDI Magnetic Media Adapter:

- 0x91xx8600 for the first drive
- 0x91xx8601 for the second drive
- 0x91xx8602 for the third drive.

(where xx indicates the slot occupied by the adapter).

Base I/O port address:

For the Personal Computer AT Fixed-Disk and Diskette Drive Adapter and ESDI Magnetic Media Adapter:

- 0x000001F0 for the primary card
- 0x00000170 for the secondary card.

For the High-Function ESDI Magnetic Media Adapter:

- 0xF00001F0 if the card is the primary card.
- 0xF0000170 if the card is the secondary card.

The High-Function ESDI Magnetic Media Adapter device driver does not support two such adapters on the same machine. If the High-Function ESDI Magnetic Media Adapter is the secondary card, the primary card must be a Personal Computer AT Fixed-Disk and Diskette Drive Adapter or ESDI Magnetic Media Adapter.

Number of I/O port addresses:

8

Bus memory start address:
Not used by these device drivers.

Bus memory end address:
Not used by these device drivers.

DMA type: Used only by the High-Function ESDI Magnetic Media Adapter:

- 0xE8000301 = primary DMA channel
- 0xE8000300 = alternate DMA channel
- 0xE8000303 = secondary DMA channel.

The adapter has the following characteristics:

- Alternate DMA supported
- Use IOCC buffering
- Only use DMA
- Use 8 DMA windows on channels 0, 1, 2.

Interrupt type:
Depends on adapter configuration

For the PC AT or ESDI adapters, this value is set to 0xB000020E and the adapters has the following characteristics:

- Interrupts enabled
- Interrupts not shareable
- Interrupt level = 14.

For the High-Function ESDI adapters alone (no standard ESDI or PC AT adapters on the same machine), this value is set to 0xE000020E and the adapter has the following characteristics:

- Interrupts enabled
- Interrupts shareable
- Interrupt level = 14.

For the High-Function ESDI adapters used with standard ESDI or PC AT adapters on the same machine, this value is set to 0xE000020C and the adapter has the following characteristics:

- Interrupts enabled
- Interrupts shareable
- Interrupt level = 12.

Device Characteristics

The device characteristics for these device drivers are structured as shown in the figure below:

0	Length			
4	Formatted Data Capacity			
8	Reserved		Interleave	Sector Size
12	Last Cylinder		Last Head	Last Sector
16	Precomp Val.	Device Status	CE Cylinder	
20	End of Life		Seek Profile	
24	Seek Profile			
28	Seek Profile			
32	Seek Profile			
36	Seek Profile			
40	Seek Profile			
44	Seek Profile			
48	Seek Profile			
52	Manufacturer ID			Reserved
56	Service Request Number		Drive Char.	Reserved

Figure 5-6. Device Characteristics

Length: Indicates the length of the device characteristics section (15 words).

Formatted data capacity:
The number of usable sectors that the drive contains. This does not include the sectors on the CE cylinder.

Interleave factor:

- High-Function ESDI Magnetic Media Adapter – 1
- Personal Computer AT Fixed-Disk and Diskette Drive Adapter – 2
- ESDI Magnetic Media Adapter – 4

Sector size: The size of sectors on the device in multiples of 256. The value for this device is 02.

Last cylinder: The last cylinder available on the drive. This is one less than the actual number of cylinders because the CE cylinder is not available for user data.

Last head: The last head available on the drive.

Last sector: The last sector available on the drive.

Precompensation value:
The starting cylinder where precompensation should be used. The value used is the cylinder number divided by four. If a fixed-disk drive does not use precompensation, this field should be set to 0xFF.

Device status: Used by the POST routine after the configuration record is read from the disk. When the configuration record is written, the value of this field is 0x00.

CE cylinder: The cylinder used for diagnostic tests. This cylinder is always the last cylinder on the fixed-disk drive and is never used for user data.

End of Life: Indicates the maximum number of sector defects that are allowed before the drive is considered to be bad.

Seek profile: The parameters used to approximate the amount of time the drive will need to perform a seek operation. This is necessary because this fixed-disk drive does not send an interrupt upon completion of a seek request. Therefore, the device driver calculates how long the seek operation should take and allows the system to perform other tasks during that time. After the time interval has passed, the device driver checks to see if the seek is actually complete.

For the High-Function ESDI Magnetic Media Adapter, this value is set to zero.

Manufacturer ID:
The identification number of the drive manufacturer.

Service Request Number:
Contains an SRN used by the diagnostics for this drive.

Drive Char.: Contains an ASCII character that, when combined with the drive capacity, identifies the fixed disk.

Error Log

The error log for these device drivers is organized as shown in Figure 5-7 on page 5-54. It is logically divided into four sections, each of which is used for a different purpose.

The first section is always used when an error occurs. It contains the standard error log information such as length, class, subclass, mask, and type.

The second section starts at byte 8 and is only used when a permanent error occurs. This is indicated in the type field with a value of 0x80. This section contains information about the current I/O operation and the device that uses this device driver.

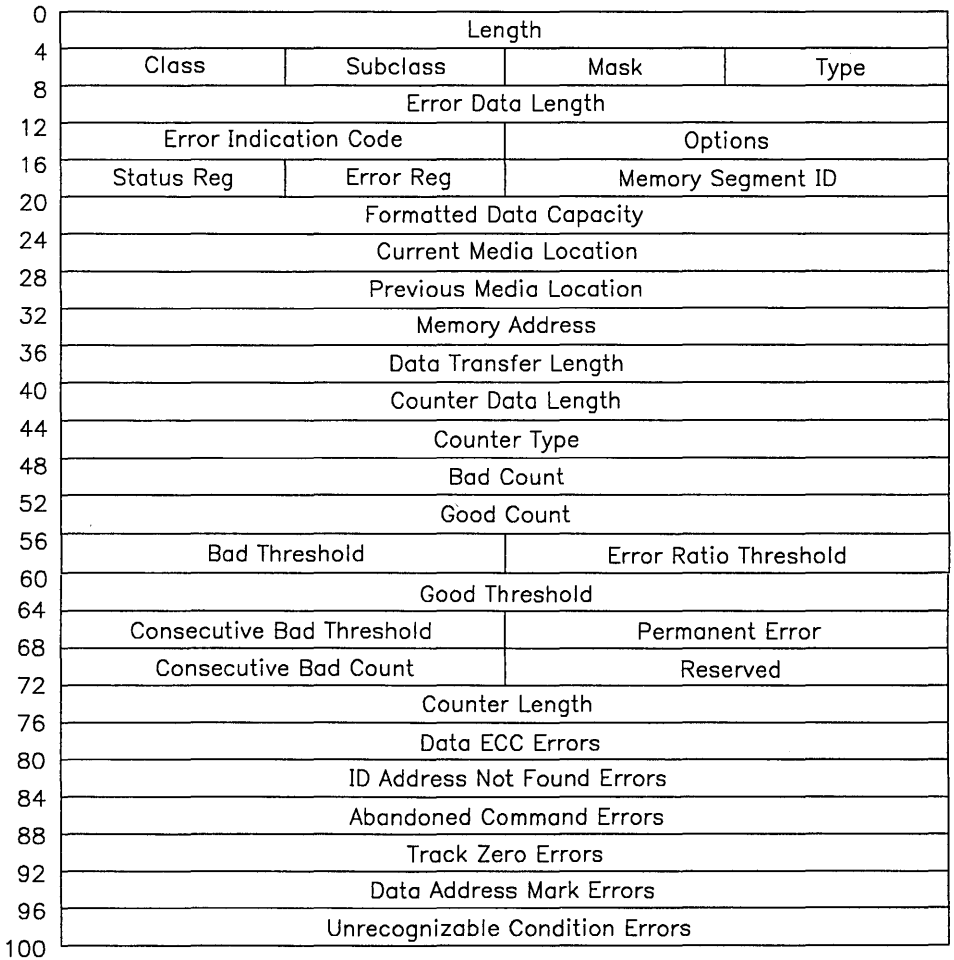


Figure 5-7. Error Log Structure

The third section starts at byte 40 and is only used when an error causes a predefined error ratio threshold to be exceeded. This is indicated in the type field with a value of 0x20. This section contains information about the error that caused the ratio to be exceeded.

The fourth section starts at byte 72 and is also only used when an error causes a predefined error ratio threshold to be exceeded. This section contains a list of counters that identify the total number of errors encountered for each type of error that can occur for this device driver.

"DDS Error Log" on page 2-6 describes how device drivers can be designed to monitor errors using this type of error log structure. You may need to read that section to understand the purpose of the fields of this error log.

The various fields of the error log are defined as follows:

Length: 25 words

Class: 0x01 for hardware errors.

0x02 for software errors.

Subclass: 0x02

Mask: 0x00

Type: Can be one of three values:

- 0x80 indicates that a permanent error occurred. This means that the second section will be used.
- 0x20 indicates that an ECC error was corrected. No error occurred, but an error entry is generated to inform the user of the event. For these types of entries, the second section is used.
- 0x10 indicates that the error ratio threshold was exceeded. This means that the third and fourth sections will be used.

Error data length:
8 words.

Error indication code:
Uses bits 9 through 15 to represent one of seven different types of errors that can occur. The bits are defined as follows:

- Bit 9 — Unrecognizeable condition
- Bit 10 — Bad data address mark
- Bit 11 — Track 0 error
- Bit 12 — Time Out error
- Bit 13 — Bad ID address error
- Bit 14 — Uncorrectable ECC error
- Bit 15 — Correctable ECC.

  Bit 15 is used to signal that the device driver was able to correct an ECC error. The device driver generates an error entry because it is a significant event that might be of interest to the error log analysis routines.

Options: The operation option and device option sent to the device driver.

Status Reg: The last status register contents sent from the adapter. More than one of the conditions may be set. They are defined below:

- Bit 0 — Adapter busy
- Bit 1 — Drive ready

- Bit 2 — Write fault
- Bit 3 — Seek complete
- Bit 4 — Data request
- Bit 5 — Corrected data
- Bit 6 — Index
- Bit 7 — Error.

Error Reg: The last error register contents sent from the adapter. It can be one of the following values:

- Bit 0 — Bad block
- Bit 1 — Data ECC error
- Bit 2 — Reserved
- Bit 3 — ID not found
- Bit 4 — Reserved
- Bit 5 — Abandoned command
- Bit 6 — Track 0 error
- Bit 7 — Data address mark not found.

Memory segment ID:
The segment register contents at the time of the error.

Maximum media length:
The maximum number of sectors on the fixed disk.

Current media address:
The physical location on the fixed disk where the error occurred. This is made up of four bytes as shown:

- Bytes 0 and 1 — Cylinder number
- Byte 2 — Head number
- Byte 3 — Sector number.

Previous media address:
The previous physical location of the fixed-disk read/write head. This is made up of four bytes as shown:

- Bytes 0 and 1 — Cylinder number
- Byte 2 — Head number
- Byte 3 — Sector number.

Memory address:
Location within the memory buffer where the error occurred.

Data transfer length:
The number of bytes that have been transferred either to or from a memory buffer.

Counter data length:
8 words

This is the length of the error counter data returned when the error ratio threshold is exceeded.

Counter type: The first bit in this field is always set to one. This field also uses bits 10 through 15 to represent six different types of errors that are monitored by this device driver. One of these bits is set to one to indicate which error caused the error ratio threshold to be exceeded. Bits 10 through 15 are defined as follows:

- Bit 10 — Unrecognizable condition
- Bit 11 — Bad data address mark
- Bit 12 — Track 0 error
- Bit 13 — Time-out error
- Bit 14 — Bad ID address error
- Bit 15 — Uncorrectable ECC error.

Bad count: Total number of errors encountered for the type of error specified in the counter type field.

Good count: Number of successful I/O operations performed. The error ratio threshold is not checked until this value reaches 40. This ensures that enough operations have been performed to obtain meaningful results from an error ratio check.

Bad threshold: Number of retries that must occur before checking the error ratio threshold. The value for this device driver is 3.

Error ratio threshold:
The value which will cause the error counter data to be reported. This value is obtained by dividing the number of errors for a particular type of error by the good count. This device driver sets the threshold at 6, meaning 6 percent.

Good threshold: The maximum number of good operations which can be used to compute the error ratio. If the good count counter reaches this value, it is reset to 0 by the device driver. This ensures that the good count value does not become so large that the error ratio becomes statistically invalid. The value for this device driver is 640.

Consecutive bad threshold:
Number of consecutive retries that must occur before an error is reported. The value for this device driver is 3.

Permanent error threshold:
Number of retries that must occur before logging a permanent error. The value for this device driver is 3. If an I/O operation reaches this value, the operation is abandoned and reported as unsuccessful.

Consecutive bad count:
Number of consecutive retries that occurred for the last I/O operation. This value can range from 1 to 3.

Counter length: 7 words.

This section is used to report the totals for each type of error when the error ratio threshold is exceeded.

Note that all of the error counters are reported in the last 6 words, even though only one particular type of error caused the counter data to be reported.

CCB Input Values

This device driver accepts CCBs as shown in Figure 5-8:

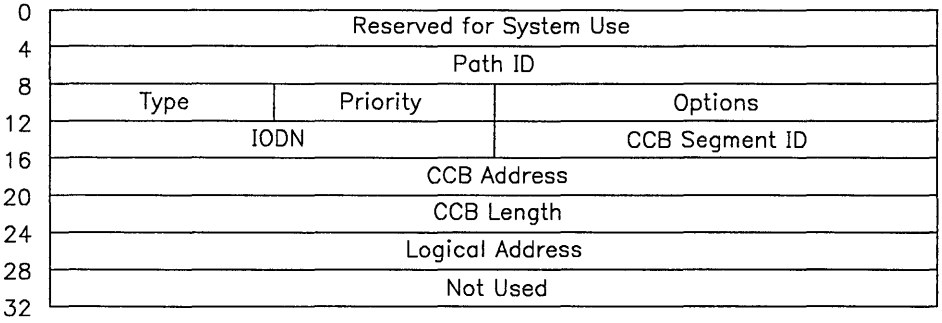


Figure 5-8. CCB Queue Element

The CCB consists of the command header and zero or more command elements, depending on the device option used. The fields specific to CCBs used by this device driver are defined below:

Type: This is set to 2 to indicate that this is a CCB queue element.

Options: Each bit is defined below:

- Bit 0 through 4 — Operation option
- Bit 5 — Bad blocks

  The minidisk manager uses this bit to tell the fixed-disk device driver that the specified request contains references to bad blocks. When this bit is set, the driver must call the minidisk manager to get alternate block information when the request is initiated. This bit cannot be set by the user.

- Bit 6 — Write verify

  The minidisk manager uses this bit to tell the fixed-disk device driver that the write operation should be followed immediately by a verify operation. If the verify detects any errors, the driver will remap the defective sectors to alternate sectors and retry the write operation. The requestor receives an error only if the remap is unsuccessful. This bit can be set only by the minidisk manager or the requestor of an internal VRM operation.

- Bit 7 — Reserved
- Bit 8 — Not used
- Bit 9 — Disable bad block mapping

  This bit can be set by a user to prevent the fixed-disk device driver from alternate sector mapping. If a request needs to be contiguous (such as data read by read-only memory), this bit should be set.

- Bit 10 — Memory addressing mode

  This bit indicates whether an address is virtual or real. For CCBs, it is always set to zero to indicate virtual address mode.

- Bit 11 through 15 — Device option

  The device options available for this device driver are defined below. Device options that require a command element contain information about the command element in the definition. If there is no information about a command element, you can assume that the device option does not use a command element.

  0 = Read

  This device option reads information starting at the sector identified by the logical address field. This device option uses a CCB element with the following field definitions:

  Data transfer length:
  Any integral multiple of 512 bytes.

  Memory address:
  Location of the data buffer that will receive the sector data.

1 = Write

This device option writes information starting at the sector identified by the logical address field. This device option uses a CCB element with the following field definitions:

Data transfer length:
Any integral multiple of 512 bytes.

Memory address:
Location of the data buffer that contains the data to be written to the sector.

2 = Position

This device option positions the read/write head at the specified logical address.

3 through 31 = Not used by this device driver.

After successfully completing any of the device options, the device driver issues a dequeue function call and waits for the next device option. The dequeue function call tells the queue manager that it can release the current queue element.

If the device driver cannot successfully complete a device option, the device driver issues a dequeue function call and reports an error to the VRM error process and virtual machine.

IODN: Note that this is a minidisk IODN, not the actual IODN for the fixed-disk drive.

CCB segment ID:
This field contains the segment ID of the address of the memory buffer.

CCB address: This field contains the offset within the segment to the start of the CCB.

CCB length: The length of the CCB that contained the original request.

Logical address:
Location within the minidisk where the data transfer should take place.

Internal VRM Queue Element Values

This device driver accepts internal VRM queue elements from three components within the VRM:

- Minidisk manager
- IBM RT PC Personal Computer AT Coprocessor Option
- Virtual memory manager.

These queue elements have the structure shown in Figure 5-9:

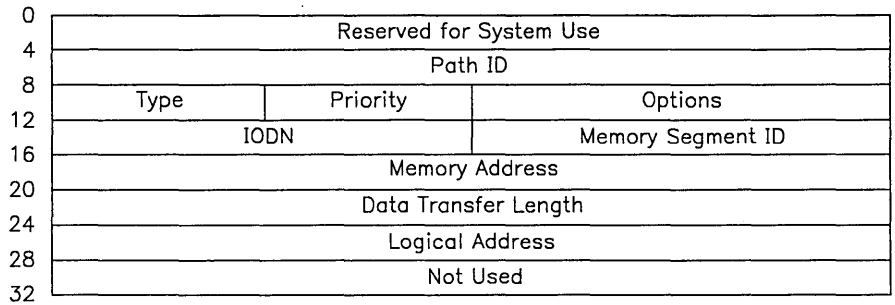


Figure 5-9. Internal VRM Queue Element

These queue elements can perform the same operations as a standard CCB issued by the virtual machine. These queue elements differ from a CCB in that there is no need for a pointer to a CCB queue element.

The fields usually defined by the CCB queue element, memory address and data transfer length, are already stored in an internal VRM queue element. Thus, these queue elements have no need for a CCB address or CCB length field. These two fields are replaced by the memory address and data transfer length fields.

The fields specific to internal VRM queue elements used by this device driver are defined below:

Type: This is set to 3 to indicate that this is an internal VRM queue element.

Options: Each bit is defined as follows:

- Bit 0 — Interrupt on completion
- Bit 1 — Interrupt on error
- Bit 2 — Synchronous operation
- Bit 3 — Command extension
- Bit 4 — Reserved for VRM
- Bit 5 — Bad block

- Bit 6 — Write verify
- Bit 7 — Reserved
- Bit 8 — Not used
- Bit 9 — Not used
- Bit 10 — Memory addressing mode

  For internal VRM requests, this bit indicates whether the address is virtual or real. A zero means the address is virtual; a one means the address is real.

- Bits 11 through 15 — Device option

  These queue elements can contain commands for the device driver in the device option portion of the options field. The commands are defined below:

  0 = Read

  This device option reads information starting at the sector identified by the logical address field. This device option requires the following fields to be set in the internal VRM queue element:

  Memory address:
  Location of the data buffer that will receive the sector data.

  Data transfer length:
  Any integral multiple of 512 bytes.

  1 = Write

  This device option writes information starting at the sector identified by the logical address field. This device option requires the following fields to be set in the internal VRM queue element:

  Memory address:
  Location of the data buffer that contains the data to be written to the sector.

  Data transfer length:
  Any integral multiple of 512 bytes.

  2 = Position

  This device option positions the read/write head at the specified logical address.

  3 through 31 = Not used by this device driver.

  After successfully completing any of the device options, the device driver issues a dequeue function call and waits for the next command.

The dequeue function call tells the queue manager that it can release the current queue element.

If the device driver cannot successfully complete a device option, the device driver issues a dequeue function call and reports an error to the VRM error process and virtual machine.

IODN: This is a minidisk IODN, not the actual IODN for the fixed-disk drive.

Memory segment ID:
If the memory addressing mode is virtual, this field contains the segment ID of the address of the memory buffer. If the memory addressing mode is real, this field is ignored.

Memory address:
If the memory addressing mode is virtual, this field contains the offset within the segment to the start of the memory buffer. If the memory addressing mode is real, this field contains the actual memory address of the memory buffer.

Data transfer length:
The total number of bytes that are to be transferred by this operation.

Logical address:
Location within the minidisk where the data transfer should take place.

Output Values

This section defines the output values that may appear in GPR2 or in a PSB. The PSB fields that contain I/O information or detected error information are the status flags field and the operation results field. Figure 2-6 on page 2-12 shows the location of these fields in the PSB.

The status flags field is a single byte and is defined as 0x24. This indicates that this is a solicited I/O interrupt.

The 16-bit operation results field returns a value indicating the completion status of the operation. The possible values that can be returned are listed below:

00 = Successful completion

256 = Initialization failed

One of the initialization routines returned an error code.

257 = Invalid device option

The CCB contained an invalid device option.

258 = Invalid data length

A request specified a data length that was not a multiple of 512 bytes.

259 = Invalid parameters

The command extension flag in the queue element was not set for a read or write operation, or the buffer address is not aligned on a word boundary.

260 = Invalid IODN

The DDS contained an invalid IODN in the DDS header section.

-02 = Timeout

The adapter would not respond to commands within the allowed time interval.

-03 = Bad block mark

The adapter detected a bad block mark in a sector ID field. The bad block mark is set during the formatting operation to identify sectors that are unusable.

-04 = ECC error

The adapter detected an uncorrectable ECC error during a read operation.

-05 = ID not found

Unable to locate the sector ID within the allowed time interval. This could be due to one of two reasons:

- The sector ID had a CRC error.
- The sector ID was not on the disk or was incorrect.

  If this condition occurs, the position of the read/write head is checked. If the head is at the wrong position, it is repositioned and the operation is tried again.

-06 = Abandoned command

Adapter stopped trying to process a command due to a detected exception condition. The exception conditions are:

- Write fault
- Incomplete seek
- Drive not ready
- Invalid command.

-07 = Track 0 not detected

A restore command did not detect track 0 after issuing 1024 step pulses to the drive.

-08 = Bad Data address mark

Unable to locate the data address mark in the data area within the allowed time interval. If this condition occurs, the position of the read/write head is checked. If the head is at the wrong position, it is repositioned and the operation is tried again.

-09 = Not used by this device driver.

-10 = Unrecognized condition

An error occurred that the device driver could not evaluate.

-11 = Bad block mapping abandoned

The device driver encountered a situation that required bad block mapping. However, bit 9 of the options field of the CCB that initiated the I/O request was set to disable bad block mapping. Therefore, the operation was abandoned.

-12 = Invalid parameters

The CCB or queue element contained invalid parameters. This could be one of the following:

- Invalid memory segment ID
- Invalid buffer address.

-40 = Subsequent read error

This minidisk manager return code is passed back through the fixed-disk device driver. This occurs if a read is attempted to a block previously notified as a bad block with no intervening write since the notification. The block in question needs to be written in order to clear the error.

Note: Corrected ECC errors are reported to the error process in the VRM; however, the return code is set to 0, indicating successful completion.

Reserved Cylinders on the Fixed Disk

Each fixed disk reserves two cylinders for special use. One of the reserved cylinders is cylinder 0, the first cylinder on any fixed disk. The disk uses the first four tracks of cylinder 0 to store various types of disk configuration and operation information. Some of this information is placed on the cylinder by the fixed-disk manufacturer, and some of it is written by the operating system. This section defines the structure and contents of the first 4 tracks of cylinder 0. If cylinder 0 contains more than 4 tracks, the additional space is used for user data.

The other reserved cylinder on the fixed disk is called the CE cylinder. This is always the last cylinder on the fixed disk and is used for diagnostic purposes. The CE cylinder cannot be used for user data.

A fixed-disk cylinder consists of the tracks that can be accessed from a specific arm position. The fixed disks supported by the VRM use 512 bytes per physical sector and at least 17 sectors per track.

The following table describes the information stored on cylinder 0 of supported fixed disks. Fixed disks that conform to the format shown here should be compatible with the operation of the VRM minidisk manager. The locations of the respective items are expressed as cylinder number (C#), head number (H#), and sector number (S#). The following formula enables you to determine the logical sector number (LSN) of the supplied information, regardless of the number of sectors per track:

```
LSN = (C# times (last head + 1) times last sector)
      + (H# times last sector)
      + S# - 1
```

The last cylinder value, last head value and last sector value are contained in the configuration record (see "Configuration Record" on page 5-69). Note that any reserved blocks or bytes are assumed to be set to 0.

Information	C#	H#	S#
IPL record	0	0	1
Configuration record	0	0	2
Reserved	0	0	3
Minidisk directory	0	0	4-6
Reserved	0	0	7-8
Bad block directory	0, 0, 9 to 0, 1, 13		
Backup configuration record	0	1	14
Backup minidisk directory	0	1	15-17
Reserved	0	1	18-n
POST control block	0	2	1-n
Reserved track	0	3	1-n

Figure 5-10. Cylinder 0 Layout

Note that logical sector numbers start at 0 and go to approximately 86,000 for a 44 megabyte fixed disk and to approximately 138,000 for a 70 megabyte disk. The sector IDs on each track are formatted 1 through the number of sectors per track, inclusive.

Each section of cylinder 0 is discussed on the following pages, with the exception of the backup configuration record and the backup minidisk directory. The format of these two backup sections is the same as its primary counterpart.

IPL Record

The IPL record consists of one block. It contains information that allows the system to read the IPL code and initialize the system. It can be divided into three logical sections. The first section is two words long and contains the IPL record ID. The second section is 6 words long and contains format information about the fixed disk. The third section is 4 words long and contains information about where the IPL code is located and the length of the code. The fields are defined below:

Bytes	Description
0-3	IPL record ID. Identifies the type of IPL record. This field is set to 0xC9C2D4C1.
4-7	Reserved.
8-9	Number of cylinders on the disk used for IPL.
10	Number of heads on the disk used for IPL.
11	Number of sectors on each track of the disk used for IPL.
12-13	Sector size as measured in bytes. For example, if the sector size was 512 bytes, this field would have a value of 0x0200.
14-18	Reserved.
19	Interleave factor.
20-23	Reserved.
24-27	Formatted disk size.
28-31	Reserved.
32-33	Cylinder containing the start of the loaded code.
34	Head number to be used to read the loaded code.
35	First sector of the loaded code.
36-39	Length of the loaded code, measured in sectors.
40-43	Offset from the start of the loaded code to the first instruction.
44-47	Block number of the VRM minidisk.
48-51	Loadlist processor block number. This is a block number that is only required for the AIX Operating System **cvid** command.
52-55	Length of VRM Minidisk. This is a block number that is only required for the AIX **cvid** command.
56-511	Reserved.

Configuration Record

The configuration record consists of one block. It contains information required by the fixed disk for I/O operations. Some of the fields in the configuration record are defined in "Device Characteristics" on page 5-52. The configuration record fields are as follows:

Bytes	Description
0-3	Configuration record ID. Identifies the configuration record as present and valid. This field is always set to 0xF8E9DACB.
4-7	Formatted data capacity.
8-9	Reserved.
10	Interleave factor.
11	Sector size.
12-13	Last data cylinder Cylinders are numbered 0 to n, where n is the value in bytes 12-13. The total number of cylinders is n + 2, where the last cylinder is the CE cylinder.
14	Last head Heads are numbered from 0 to n, where n is the value in byte 14. The total number of heads is n + 1.
15	Last sector Sectors are numbered 1 to n, where n is the value in byte 15. The number of sectors per track is n.
16	Precompensation value.
17	Device status. Used by the loadable POST routine after the configuration record is read from the disk. When the configuration record is written, the value of this field is 0x00.
18-19	CE cylinder.
20-21	End-of-life. The value in this field determines the number of defects that force a fixed disk to be considered unusable. A value of zero allows the system to set this value; a value of all ones disables bad block relocation for the disk.
22-51	Seek profile.
52-54	Manufacturer ID. Bytes 52 and 53 of this field indicate the size of the fixed disk (40 megabyte, 70 megabyte, and so on). Byte 54 indicates the ID of the manufacturer. ID values from 0x00 through 0x7F are reserved for IBM-supported manufacturers and are included

in Diagnostic Control Program tests. ID values greater than 0x7F are not checked by IBM diagnostics.

55 Reserved.

56-57 Service request number

This field contains a value that is used by the diagnostics facility to identify the disk drive.

58 Drive characteristic

This field contains an ASCII character that, combined with the drive capacity, identifies the fixed disk.

59-511 Reserved.

Minidisk Directory

The minidisk directory consists of three blocks. It is used by the VRM to describe how the fixed disk is divided into minidisks. It is initially set to all zeroes by the fixed-disk manufacturer. It is logically structured as an array of 48 entries, each of which is 32 bytes long.

The first entry, with an index of 0, is the header and contains the following:

Bytes	Description
0-3	The number of minidisks, including free space minidisks.
4-7	Level identifier of the minidisk directory.
8-9	Index of next unused directory entry.
10-11	Reserved.
12-13	Index of the first minidisk directory entry on the list of defined minidisks.
14-15	Index of the last minidisk directory entry on the list of defined minidisks.
16-19	Logical sector number of the start of the area reserved for bad block relocation.
20-23	Size of the bad block area in sectors.
24-27	I/O counter indicating the directory most recently used.
28-31	Reserved.

The remaining entries contain definitions of minidisks, which includes data minidisks and free space minidisks. Each entry in the list is defined as follows:

Bytes	Description
0-1	Index of the previous entry on the list of defined minidisks.
2-3	Index of the next entry on the list of defined minidisks.
4-7	Minidisk name from create minidisk.
8-11	Reserved.
12-15	Date the minidisk was created. This is represented as the number of seconds that have elapsed since 1970.
16-17	Minidisk IODN or 0 if it is a free space minidisk.
18	The minidisk logical block size where 0 = 512 bytes, 1 = 1024 bytes, and 3 = 2048 bytes. This value is stored in bits 0-3 of the byte. Bits 4-7 are reserved.
19	The minidisk type. This is determined by which bit is set. The bits are defined as follows: Bit 0 — Write verify Bit 1 — No bad block relocation for this minidisk Bit 2 — Paging space minidisk Bit 3 — AIX Operating System file system minidisk Bit 4 — AIX Operating System minidisk Bit 5 — IBM RT PC Personal Computer AT Coprocessor Option minidisk Bit 6 — VRM minidisk Bit 7 — Auto IPL minidisk.
20-23	Reserved.
24-27	Logical sector number for the start of the minidisk.
28-31	Number of sectors used by the minidisk.

Backup Minidisk Directory

The backup minidisk directory is used after an error has occurred during input/output operations to the primary minidisk directory. If errors are found on both the primary and backup minidisk directories during VRM initialization, the fixed disk that contains those directories will support **no** minidisk operations.

If no errors are found, the minidisk directory with the greater value in the fullword at offset 24 is used.

Bad Block Directory

The bad block directory consists of 5 + (last sector number) blocks. This directory keeps a record of the blocks that have been diagnosed as unusable.

Each entry in the bad block directory is 8 bytes long. The first entry shows that this is the bad block directory. It contains the ASCII letters **DEFECT** (0x444546454354) in the first 6 bytes followed by 2 bytes that identify the total number of bad block entries contained in the bad block directory.

The remaining entries identify specific bad blocks. Each entry is structured as shown below:

Bytes	Description
0	Reason the block was marked as unusable. This can be one of three values: 0x00 = The disk manufacturer found a defect. 0xAA = The surface verification diagnostic test found a defect. 0xBB = The system found a defect. 0xCC = The manufacturing test found a defect.
1-3	Logical sector number of the bad block. This is a 24-bit unsigned number.
4-7	Initialized to zero by the manufacturer, but subsequently filled in by the system with the logical sector number to which the bad block is relocated. A 1 in the most significant bit indicates that the data in the relocated block is invalid.

POST Control Block

One track is reserved for this section. The track contains a copy of the POST control block that is created in memory by POSTs and LPOSTs during initialization. The last LPOST is specifically designed to write the POST control block onto the third track of cylinder 0.

Graphics Asynchronous Device Driver

The graphics asynchronous device driver supports the IBM 5080 Peripheral Adapter. Devices supported by this adapter include:

- IBM 5080 Dials Feature
- IBM 5080 Lighted Program Function Keyboard Feature

The IBM 5080 Peripheral Adapter card number must be set to 4. Interrupt level 11 is recommended for use with this driver, but you can also use levels 9 and 10. The dials and lighted PF keys can be used only on ports 1 and 2 of the adapter, and only 1 of each device is allowed.

Define Device Header

Pertinent fields in the DDS header for the Graphics Asynchronous Device Driver include:

IOCN: 0x260

IODN:
- 0x39 (Dials)
- 0x3A (Lighted PF keys)

Hardware Characteristics

Hardware characteristics for this adapter are defined as follows:

Length: 0x0000000F

Internal Device Type:

- I/O bus device
- Switchable to coprocessor
- 8-bit device
- 2 interrupt level definitions
- Adapter type = 0x38

Base I/O port address:

- Primary port = 0x00004230
- Secondary port = 0x00004238

Number of I/O port addresses: 8

Bus memory start/end address: N/A

DMA type: Not supported

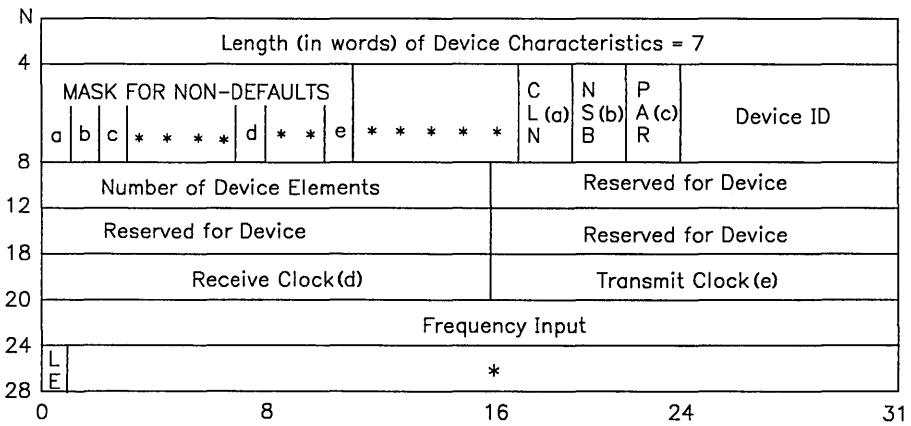
Interrupt definition:

- Interrupts enabled
- Interrupts shareable
- Interrupt levels: 0x0B (recommended), but 0x09 and 0x0A will work.

  The interrupt level applies to all ports on the adapter card.
- Device class: 3

Device Characteristics

The graphics asynchronous device driver device characteristics fields are shown in Figure 5-11.



* - Reserved (not used)

Figure 5-11. Graphics Asynchronous Device Characteristics

The parameters shown in the preceding figure apply to both the IBM 5080 Dials Feature and the IBM 5080 Lighted Program Function Keyboard Feature. The device characteristics fields are defined as follows:

- Length

  The length of the device characteristics field for this driver is 7 words.

- Mask for non-defaults

  These bits serve as flags for other device characteristics values you can specify. A 0 in any of the fields a through e indicates that you want to use the default value for the corresponding field. A 1 in any of these bits means that you will supply a valid value for the corresponding field. For example, if you place a 1 in bit 0 ('a' in the preceding figure) of the mask area, you must supply a value for the corresponding field. In this case, the corresponding field is the character length field, bits 17-18 of the second word.

- CLN (character length)

  This 2-bit field indicates character length. Possible values are:

  00 = 5 bits per character

  01 = 6 bits per character

  10 = 7 bits per character (default)

  11 = 8 bits per character

  In the preceding figure, 'a' is the mask bit for this field. If 'a' is 0, the 7 bits-per-character default is used. If a is 1, select from the values defined above for this field.

  The device driver expects to receive transmitted data right-justified in the byte with non-significant, high-order bits set equal to zero. Received data characters are also right-justified with non-significant high-order bits set equal to zero by the device driver. All data is sent or received with the least-significant bits first.

- NSB (number of stop bits)

  This 2-bit field defines the number of stop bits. Possible values are:

  00 = Reserved

  01 = 1 stop bit (default)

  10 = 1.5 stop bits

  11 = 2 stop bits

  'b' represents this field's mask bit in the preceding figure.

- PAR (parity checking and generation)

  This 3-bit field defines the parity checking (Rx) and generation (Tx) capabilities of this driver. Possible values are:

  000 = No parity

  001 = Odd parity

  010 = Mark parity

011 = Even parity (default)

100 = Space parity

101 = Reserved

110 = Reserved

111 = Reserved.

'c' represents the bit mask for parity checking in the preceding figure.

- Device ID

  This field indicates the device type attached to the adapter. Possible ID values are:

  0x00 = Device bad or not present

  0x63 = IBM 5080 Lighted Program Function Keyboard Feature

  0x64 = IBM 5080 Dials Feature

- Number of Device Elements

  This field defines the number of input device elements. For the IBM 5080 Lighted Program Function Keyboard Feature, this field indicates the number of keys on the keyboard. For the IBM 5080 Dials Feature, this field indicates the number of dials.

- Reserved for Device

  These three halfwords are reserved for graphics data provided by the device.

- Receive clock

  This halfword contains the receive data baud rate for the adapter. Because the receive and transmit rates for supported adapters must be the same, this value must be set equal to the value of the transmit clock.

  The default value for this field is 9600 bits per second (bps), and 'd' represents the default bit mask in the preceding figure. See Figure 5-12 on page 5-77 for valid data rates.

- Transmit clock

  This halfword contains the transmit data baud rate for the adapter. The value in this field must match the value set for the receive clock.

  The default value for this field is 9600 bps, and the default bit mask is represented by 'e' in Figure 5-11 on page 5-74. See Figure 5-2 for valid data rates.

- Frequency input

  This field contains timer input to the adapter interface. For the IBM 5080 Peripheral Adapter, the value is 1.8432 megahertz (Mhz). The hexadecimal value for this field is 0x001C2000.

- LE (LPOST error)

  When this bit is set, the LPOST for the IBM 5080 Peripheral Adapter detected an error at IPL time.

Valid Clock Values	Data Rates
0x0000	50
0x0001	75
0x0002	110
0x0003	134.5
0x0004	150
0x0005	300
0x0006	600
0x0007	1200
0x0008	1800
0x0009	2000
0x000A	2400
0x000B	3600
0x000C	4800
0x000D	7200
0x000E	9600
0x000F	19200

Figure 5-12. Graphics Asynchronous Data Rates

Error Log

The graphics asynchronous device driver logs only temporary errors. The format of the error log for this device driver is similar to the log shown in Figure 5-21 on page 5-118. The fields defined for the graphics asynchronous error log include:

Length: 6 words

Class: Both hardware (0x01) and software (0x02) errors are logged.

For hardware errors (0x01), the following values apply:

Subclass: 0x05 (display station)

Mask: 0x03 (5080 Peripheral Adapter)

Error Data Length: 4 words

For software errors (0x02), the following values apply:

Subclass: 0x07 (VRM component)

Mask: 0x05 (5080 Peripheral Adapter software)

Error Data Length: 4 words

Error Data for the graphics asynchronous error log consists of three additional words of data. These words are defined as follows:

Graphic Device ID:
This field indicates the device for which the error is logged. Possible values include:

- 0x63 for the Lighted Program Function Keyboard Feature
- 0x64 for the IBM 5080 Dials Feature.

Current Command:
The next word indicates the command that was being processed at the time of the error. Possible values include:

- 0x00 = no command active
- 0x64 = output data command active
- 0x65 = current virtual terminal command active.

Error Value: This word provides more detail on the specific error logged. Possible values include:

- 0x01 = software buffer overflow (for class 0x02 software errors only)
- 0x02 = hardware receiver overrun
- 0x04 = received parity error
- 0x08 = received framing error
- 0x10 = received break signal detected
- 0x100 = bad acknowledgment from IBM 5080 Dials Feature
- 0x200 = received data sequence error from Lighted Program Function Keyboard Feature
- 0x400 = transmission timeout failure.

Input Values

The graphics asynchronous device driver receives **Send Command SVC** instructions from a virtual machine. The device options field of the operation options halfword in the Send Command queue element indicates the requested function. Two functions are supported. They are:

- Output data (device option 0x64)

  This device option directs the device driver to output data in the command extension. If the 'number of bytes to send' field is set to zero, the data in the command extension is neither validated nor sent to the device.
- Current virtual terminal (device option 0x65)

  This device option informs the device of the current virtual terminal and provides the ID of the device-to-current virtual terminal path.

  This command also directs the device driver to output the data in the command extension. This is useful for setting new device characteristics at the same time that the path to the new virtual terminal is established. If the 'number of bytes to send' field is zero or if the current virtual terminal path ID is zero, the data in the command extension is neither validated nor sent to the device.

Required queue element fields for the valid device options are defined as follows:

- Synchronous - the command must be enqueued to the device driver as a synchronous request.
- Command extension fields

  Both device options require command extension buffers. Therefore, both requests must fill in the command extension segment ID field of the Send command queue element (byte offset 14). Also required are the command extension address (byte offset 24) and command extension length (byte offset 28).
- Number of bytes to send

  Both the output data and current virtual terminal requests must indicate the number of bytes to send with the request. This value is contained in byte offset 16 of the Send Command queue element. If this value is zero, the data is neither validated nor sent to the device.
- Path ID

  For the current virtual terminal request, byte offset 20 contains the ID of the device driver-to-virtual terminal path. For the current virtual terminal command, a zero in this field causes the data to be neither validated nor sent to the device. For the output data command, this field is not used.

The Send Command queue element is shown in *VRM Programming Reference.*

Output Values

The check parameters routine of the graphics asynchronous device driver can generate the following return codes:

- 0 = successful
- -2 = transmission failed - timeout (returned from I/O initiation)
- 256 = invalid device option (any option other than 0x64 or 0x65)
- 257 = invalid device data

  Valid data for the LPF keys consists of a single Set Indicator command. For the dials, valid data consists of a single Read Dials command, or up to 16 Set Granularity commands.
- 261 = no command extension

When the device driver is successfully attached and a valid current virtual terminal command (with a non-zero path ID specified in byte offset 20) is received, the driver is able to send received data from attached devices to the current virtual terminal. When the driver receives valid input data from a device, a time stamp is appended to the data. The first three bytes of the time stamp contain the time in seconds since IPL of the machine. The last byte contains the current count of the VRM 60 Hz counter. The data to be returned (including the time stamp) is placed in a general purpose queue element and enqueued to the current virtual terminal using the path ID supplied in the last current virtual terminal command. *VRM Programming Reference* shows the format of a general purpose queue element. Data specific to this driver includes:

- Path ID - from the last current virtual terminal command
- Options - the first byte of the Options field indicates the interrupt options. The second byte indicates the device type.
  0x63 = LPF keys
  0x64 = Dials
- Offset 14 - this halfword contains the dials delta (for dials) or 0 (for LPF keys)
- Offset 16 - this halfword contains the LPF key number or dial number. Note that the first key or dial is number zero, the second is number one, and so on.
- Offset 20 - this fullword contains the time stamp.

The bytes at offsets 9, 12, 13, 18, 19, and 24 through 31 are reserved (set equal to zero).

Parallel Device Driver

This device driver supports a parallel communication port. In this system, the parallel port sends data to a printer. This device driver supports the following adapter cards:

- The parallel port of the IBM Monochrome Display and Printer Adapter
- The parallel port of the IBM Personal Computer AT® Serial/Parallel Adapter.

You can install two Personal Computer AT Serial/Parallel Adapter cards or one Personal Computer AT Serial/Parallel Adapter and one IBM Monochrome Display and Printer Adapter. You cannot install two IBM Monochrome Display and Printer Adapters because they would both have the same base I/O port address and interrupt type.

The Personal Computer AT Serial/Parallel Adapter uses a jumper that allows each parallel port on the adapter to have a unique base I/O port address and interrupt type. This is explained in greater detail in "Hardware Characteristics" on page 5-83 and in the documentation provided with the adapter cards.

The main point to remember is that the base I/O port address and interrupt type field values can vary on Personal Computer AT Serial/Parallel Adapters. An IBM Monochrome Display and Printer Adapter does not use a jumper; its base I/O port address and interrupt type field values are always the same.

When you have two parallel ports installed in the system, each parallel port has its own copy of the parallel device driver and its own DDS. This is a non-shared device driver. Therefore, only one virtual machine can be attached to this device driver at a time. However, if you have two parallel ports, you can attach a separate virtual machine to each port.

The parallel ports are not preconfigured devices like the diskette, display, and keyboard. Therefore, their DDSs must be created by the operating system. This is handled by the configuration routines. The parallel device driver is defined on the following pages.

Define Device Header

This section of the DDS is common to all device drivers. The information is defined as follows:

IODN: Set by the virtual machine at initialization time.

IOCN: Set by the virtual machine at initialization time.

Define option: 0x0001

Device type: 0x0001

Device name: This field is ignored by the VRM. It can be used to store a name for the device that uses this device driver.

Offset to hardware characteristics:
28

Offset to device characteristics:
72

Offset to error log:
80

Hardware Characteristics

The values for the fields in this section are predefined for the IBM Monochrome Display and Printer Adapter. For the Personal Computer AT Serial/Parallel Adapter, the base I/O port address and interrupt type fields can be one of two values.

The position of a jumper on the Personal Computer AT Serial/Parallel Adapter determines which values apply to the parallel port on the Personal Computer AT Serial/Parallel Adapter. When you have two parallel ports installed in the system, this jumper must be set correctly to prevent both ports from having the same base I/O port address and interrupt type.

The various combinations are defined in the field descriptions below:

Length: 11 words

Internal device type:

- 0x91002300 for an Personal Computer AT Serial/Parallel Adapter
- 0x91004900 for an IBM Monochrome Display and Printer Adapter.

Base I/O port address:

- 0x000003BC for the IBM Monochrome Display and Printer Adapter.
- 0x00000278 for the Personal Computer AT Serial/Parallel Adapter when used in conjunction with an IBM Monochrome Display and Printer Adapter.

  If you have a second Personal Computer AT Serial/Parallel Adapter installed, the first one is 0x00000278 and the second one is 0x00000378.

  If a single Personal Computer AT Serial/Parallel Adapter is installed, and no IBM Monochrome Display and Printer Adapter is installed, it can use either of the two addresses for that adapter.

Number of I/O port addresses:
3

Bus memory start address:
Not used by this device driver.

Bus memory end address:
Not used by this device driver.

DMA type: Not used by this device driver.

Interrupt type:

- 0x80000305 if the Base I/O Port Address is 0x00000278
- 0x80000307 if the Base I/O Port Address is 0x000003BC or 0x00000378.

This means that:

- Interrupts are enabled.
- Interrupts are not shareable.
- This device does not overrun.
- Interrupt level = 5 or 7, depending on the base I/O port address.

Device Characteristics

The device characteristics section is two words long. The fields are defined as follows:

Length: This field is one word long and is set to 2.

Characteristics:
This field is one word long and uses bits 3 through 7 of the fourth byte to define the device characteristics. Each bit represents a specific device characteristic. If a bit is set to one, the corresponding device characteristic is activated. The bits are defined below:

- Bit 0 — Not used
- Bit 1 — Not used
- Bit 2 — Not used
- Bit 3 — (IE) Enable acknowledge interrupts
- Bit 4 — (SL) Select printer
- Bit 5 — (IT) Set initialization line inactive
- Bit 6 — (AF) Set auto line feed on
- Bit 7 — (ST) Set strobe active.

For printing, the bits are set to 00011100. For initialization, the bits are set to 00001000.

Error Log

The Error Log section for this device driver is structured as shown in the following figure.

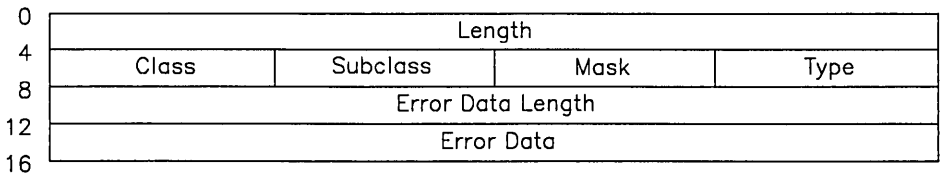


Figure 5-13. Error Log Structure

The fields specific to this device driver are defined as follows:

Length: 4

Class: 0x01

Subclass: 0x09

Mask:

- 0x01 if the parallel port is on a Personal Computer AT Serial/Parallel Adapter.
- 0x02 if the parallel port is on an IBM Monochrome Display and Printer Adapter.

Type: 0x80

Error Data Length: 2

Error Data: This field uses the third and fourth byte of the error data word. The third byte indicates the general type of error. It can be one of three values:

- 0x01 — Printer timeout
- 0x02 — Printer error
- 0x03 — Adapter initialization error.

The fourth byte defines the specific error by setting a bit:

- Bit 0 — Printer busy line is active
- Bit 1 — Acknowledge line is active
- Bit 2 — Paper end line is active
- Bit 3 — Printer is not selected
- Bit 4 — Printer error line is active
- Bit 5 — Invalid CCB
- Bit 6 — Initialization failed
- Bit 7 — Timeout occurred.

CCB Input Values

This device driver accepts CCBs as shown in Figure 2-5 on page 2-12. The CCB consists of the command header and zero or more command elements, depending on the device option used.

This device driver uses the following definitions for the bits in the options field of the CCB. Note that this device driver uses eleven bits for device options whereas other device drivers use only five bits for device options.

- Bit 0 through 4 — Operation option
- Bit 5 through 15 — Device option.

There is one additional field in a parallel device driver CCB called the characteristics field. It is located at byte offset 11 from the beginning of the CCB header and is one byte long. The characteristics field uses bits 3 through 7 of the field to define the device characteristics. Each bit represents a specific device characteristic. If a bit is set to one, the corresponding device characteristic is activated. The bits are defined below:

- Bit 0 — Not used
- Bit 1 — Not used
- Bit 2 — Not used
- Bit 3 — (IE) Enable acknowledge interrupts
- Bit 4 — (SL) Select printer
- Bit 5 — (IT) Set initialization line inactive
- Bit 6 — (AF) Set auto line feed on
- Bit 7 — (ST) Set strobe active.

The device options are defined below. Device options that require a command element contain information about the command element in the definition. If there is no information about a command element, you can assume that the device option does not use a command element.

0 = Not used by this device driver

1 = Write

This device option writes data to the printer. There must be at least one command element with each write device option. The command elements define the system data buffers that contain the data to be sent to the printer. The characteristics field must also be set to 00011100.

2 = Initialize

This option initializes the printer. The characteristics field must also be set to 00010000.

3 = Not used by this device driver.

4 = Change characteristics

This device option changes the characteristics of the printer. This device option requires the characteristics field to be set to the proper values. If a bit is set to one, the corresponding device characteristic is activated. The bit definitions are shown below:

- Bit 0 — Not used
- Bit 1 — Not used
- Bit 2 — Not used
- Bit 3 — (IE) Enable acknowledge interrupts
- Bit 4 — (SL) Select printer
- Bit 5 — (IT) Set init line inactive
- Bit 6 — (AF) Set auto line feed on
- Bit 7 — (ST) Set strobe active.

5 through 7 = Not used by this device driver

8 = Discard current queue element

This is another device option used with the check parameters routine. The virtual machine sends this device option to the device driver if it is unable to clear the condition that caused the error. This device option instructs the device driver to have the queue manager discard the current queue element from the queue. One of the following return codes will be synchronously sent back to the virtual machine:

311 — No current queue element.

312 — Current queue element discarded.

9 = Not used by this device driver.

10 = Resume print operation

This device option is used when an error of some sort has occurred and the device driver has sent an enqueue function call to generate an interrupt. After receiving the interrupt, the virtual machine examines the PSB and decides how to handle the interrupt. The virtual machine then issues a check parameters call to the device driver.

The check parameters call uses a new CCB to carry a command from the virtual machine to the device driver. This command is stored in the device option portion of the Options field of the CCB. The device driver reads the CCB and attempts to perform the command indicated.

In this case, the device option is 8, which means to resume the print operation, if possible. The device driver attempts to perform the device option and sends back one of the following return codes:

330 — Successfully resumed printing operation.

331 — Unable to resume print operation.

When the virtual machine gets this return code, it may either try another resume print operation device option or it can try to dequeue the current queue element. This is explained in the definition of device option 8.

11 through 31 = Not used by this device driver.

After successfully completing any of these device options, the device driver issues a dequeue function call and waits for the next device option. The dequeue function call tells the queue manager that it can release the current queue element and also generates an interrupt to the virtual machine.

If the device driver cannot successfully complete a device option, the device driver issues an enqueue function call, which generates an interrupt request to the virtual machine. The current queue element is retained for further processing.

Output Values

This section defines the output values that may appear in GPR2 or in a PSB. The PSB fields that contain I/O information or detected error information are the status flags field and the operation results field. Figure 2-6 on page 2-12 shows the location of these fields in the PSB.

The status flags field is a single byte and is defined as follows:

- Bits 0 through 4 are always set to 00100. This indicates an I/O interrupt.
- Bit 5 is the only variable bit. If Bit 5 is set to one, the interrupt was solicited. This means that the virtual machine was expecting this interrupt and that the I/O operation was successfully completed.

  If Bit 5 is set to 0, the interrupt was unsolicited. The virtual machine must then look at the operation results field to try to find out why the interrupt occurred.
- Bits 6 and 7 are always set to 0.

The 16-bit operation results field returns a value indicating the completion status of the operation. If Bit 0 of the first byte is set to 0, the operation was successful. If it is set to 1, an error of some sort occurred.

The second byte indicates the specific type of error that occurred. This is shown by setting one of the bits. Each bit corresponds to a different type of error. They are defined as follows:

- Bit 0 — Printer busy line is active
- Bit 1 — Acknowledge line is active
- Bit 2 — Paper end line is active
- Bit 3 — Printer is not selected
- Bit 4 — Printer error line is active
- Bit 5 — Invalid CCB
- Bit 6 — Initialization failed
- Bit 7 — Timeout occurred.

Small Computer Systems Interface Device Driver

This device driver supports the RT PC Small Computer Systems Interface Adapter for IBM 9332 DASD Attachment. The Small Computer Systems Interface (SCSI) device driver can run as many as seven 9332 DASD Attachments (at a maximum of 400M bytes per attachment), providing up to 2800M bytes of additional disk file space.

The SCSI device driver supports up to 28 9332 DASD attachments (2 SCSI adapters at 14 possible attachments per adapter). The 9332 DASD Attachments are considered external disks, which means that no IPLable minidisks can be placed on them. The disk space provided by the 9332 DASDs is intended for application programs and user space.

Figure 5-14 shows some 9332 DASD Attachments connected to a SCSI adapter and introduces several points about how these devices function in the RT PC environment.

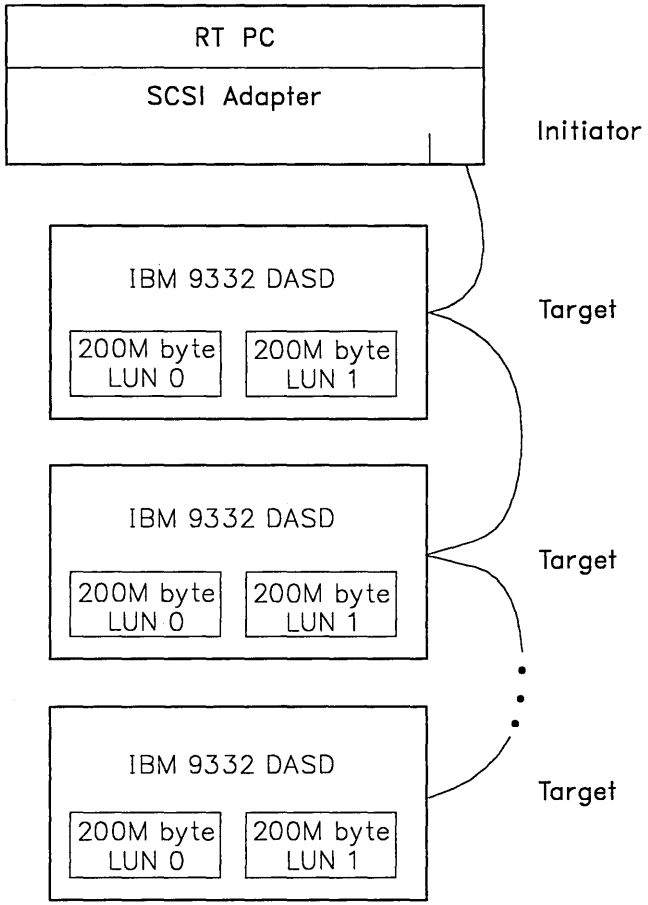


Figure 5-14. SCSI Adapter Supporting 9332 DASD Attachments. As many as seven 9332 DASD Attachments can be supported by a single SCSI adapter.

The SCSI adapter serves as an initiator of operations to the target devices attached to it. Although each 9332 DASD Attachment has its own controller, these devices merely respond to the commands of the initiator (in this case, the SCSI adapter). Because each SCSI adapter is by default an initiator, SCSI adapters cannot be used for communication between two or more RT PCs.

Each SCSI component is identified by a SCSI ID and a logical unit number (LUN). Each SCSI adapter supports eight SCSI IDs, including the ID of the adapter itself, which is always ID 7. SCSI devices, such as 9332 DASD Attachments, are assigned IDs in the range zero through six.

Figure 5-14 shows how LUNs are used to identify the disk units of the 9332 DASD Attachments. The 9332 DASD Attachment is a single device that provides either 200M bytes or 400M bytes of disk file space, depending on whether one or two 200M byte units are installed in it. The SCSI ID of the 9332 DASD and the LUN of the disk unit make up the device's IODN. If two disk units are installed in a 9332 DASD, that 9332 DASD would have two IODNs associated with it.

Any valid IODN can be selected by the system, but the port number as found in byte 4 of the internal device field of the DDS hardware characteristics section indicates the SCSI ID and LUN.

Microcode on the SCSI adapter takes the commands from the device driver and routes them to the appropriate device attached to the adapter. Each SCSI adapter has 64K bytes of memory used for system DMA **read** and **write** operations. This 64K-byte area is divided into 128 (0-127) buffers of 512 bytes each.

When using two SCSI adapters on the same machine, each adapter must use a different DMA channel.

Define Device Header

This section of the DDS is common to all device drivers. The information is defined as follows:

IODN: This field contains a value used by the input/output subsystem to reference the device for I/O activity. For the primary adapter, this value is assigned by the operating system. For a secondary SCSI adapter, the value is assigned by the operating system or by the caller.

IOCN: This field contains a value that associates the DDS with the appropriate device driver.

Define Option:
0x0001

Device Type: 0x0003

Device Name: This field is ignored by the VRM. It can be used to store a name for the device that uses this device driver.

Offset to Hardware Characteristics:
28 bytes

Offset to Device Characteristics:
72 bytes

Offset to Error Log:
132 bytes.

Hardware Characteristics

This section of the DDS is common to all device drivers. The information is defined as follows:

Length: 11 words.

Internal device type:
0xA1xx5Byz

(where xx indicates the slot occupied by the adapter, y indicates the SCSI ID, and z indicates the LUN).

Base I/O port address:
0x00000D50 for the primary adapter

0x00000950 for the secondary adapter.

Number of I/O port addresses:
16.

Bus memory start address:
Not used by this device driver.

Bus memory end address:
Not used by this device driver.

DMA type: 0xEC0004xx

(where 'xx' is the DMA channel).

Channels 0, 1, 3, 5, 6, and 7 are supported. For systems with multiple SCSI adapters, note that each SCSI adapter must have a separate DMA channel assignment.

- Device supports system DMA
- Uses IOCC buffering
- Nonshared channel, only use DMA
- Channel enabled, scatter/gather not supported.

Interrupt type:
0x4000020C

- Interrupts not enabled
- Interrupts shareable
- Class = 1
- Interrupt level = 0x0C.

Device Characteristics

The device characteristics section of the DDS contains device driver parameters that indicate how the device is controlled. This section is defined as follows for the SCSI device driver:

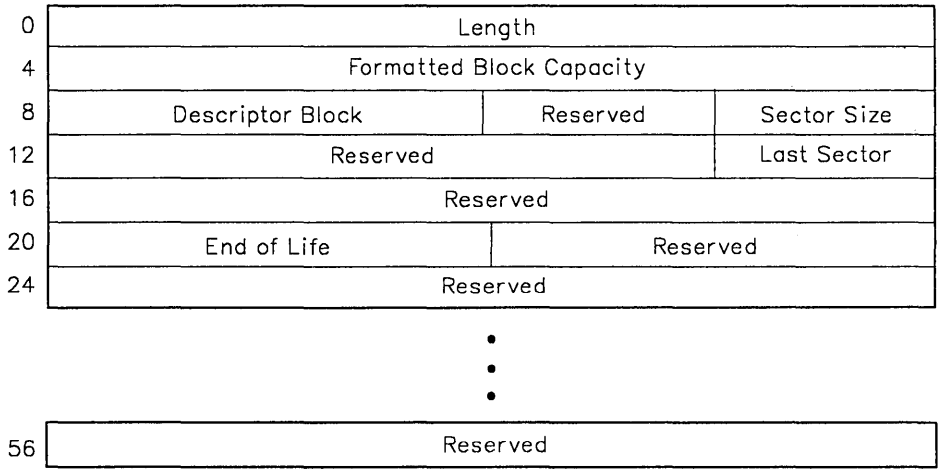


Figure 5-15. SCSI Device Characteristics

The fields in the preceding figure are defined as follows:

Length: 15 words

Formatted Block Capacity:
0x0005F80E for 9332 DASD Attachments

Descriptor Block:
This block describes the generic device type as defined by the ANSI standard for SCSI devices. The bits in this byte have the following meanings:

Bit 0: Set to one for adapters, set to zero for devices
Bit 1: Set to one for direct access storage devices (such as 9332 DASD Attachment)
Bit 2: Reserved
Bit 3: Set to one if the device is a tape drive
Bits 4-5: Reserved
Bit 6: Device stops on error (set for 9332 DASD). If an error is detected, the SCSI device terminates the operation without transferring additional data beyond the error.

Bit 7: Set to one if the device type is other than those described in bits 1-5.
Bit 8: Reserved
Bit 9: Non-negotiate bit. This bit controls whether adapter-to-device data transfer is synchronous (bit set to 0) or asynchronous (bit set to 1).
Bits 10-15: Reserved.

Sector Size: Set to two for 9332 DASD Attachments (for 512-byte blocks), or to some multiple of 512.

Last Sector: The operating system sets this field to indicate the last sector on the track for disk or direct-access devices. The default value is 17.

End of Life: The operating system sets this field to indicate the number of bad block errors that will be corrected by the system before the disk will be considered unusable. The default value is 600.

Error Log

The error log section for this device driver is structured as shown in the following figure.

0	Length			
4	Class	Subclass	Mask	Type
8	Error Length			
12	Error Indication		Operation Options	
16	Status Reg.	Error Reg.	Memory Segment ID	
20	Device Type	Reserved		
24	Current Logical Address			
28	SCSI ID	LUN	Reserved	
32	Memory Address			
36	Data Length			
40	Device Error Log Length			
44	Error Log			
48	Error Log			
52	Error Log			
56	Reserved			

Figure 5-16. SCSI Error Log Structure

The fields in the preceding figure are defined as follows:

Length: 15 words

Class: 0x01 to indicate hardware error.

Subclass: 0x0E

Mask: 0x01

Type: Can be one of the following values:

- 0x08 is an error with the processor on the 9332 DASD Attachment.
- 0x10 or 0x11 is an error with the file processor.
- 0x20 or 0x40 is an error with the servo processor.
- 0x80 is an error from the adapter.
- 0xC0 is an error from another SCSI device.

Error Length: Indicates the length of the header information. For this device, set to eight words.

Error Indication: The first byte of this halfword is a copy of the Type field in byte offset 7. The second byte is zero for 0x80 and 0xC0 errors. For all other error types (0x10, 0x11, 0x20, 0x40), the second byte contains an error value from the 9332 DASD Attachment.

Operation Options:
This halfword is a copy of the Operation Options field of the request that caused the error.

Status Reg.: This byte contains the contents of the adapter's status register.

Error Reg.: This byte contains the contents of the adapter's error register.

Memory Segment ID:
This field indicates the segment ID of the data buffer from the request that caused the error.

Device Type: This byte is a copy of the Descriptor Byte field of the device characteristics section.

Current Logical Address:
This field indicates the logical address for the failing **read** or **write** request.

SCSI ID: Indicates the port address for the device that failed on this command. For the SCSI adapter, this value is always seven. Possible values for other SCSI devices are zero through six.

LUN: This field indicates the logical unit number associated with the SCSI ID where the error occurred. For example, a 9332 DASD Attachment with two 200M byte disk units would have two LUNs associated with its SCSI ID, zero and one. Possible values are zero through eight.

Memory Address: This field indicates the location of the memory buffer.

Data Length: This field indicates the length of the failing command.

Device Error Log Length:
This field indicates the length of the error log for the device. For the 9332 DASD, the value is five (one word for length, three words of error log data, and one reserved word).

Input Values

The SCSI device driver accepts requests both from virtual machines (by way of **Start I/O SVCs**) and from such VRM components as the minidisk manager, coprocessor, and virtual memory manager (by way of general purpose queue elements).

Requests can be divided into two general groups, input/output requests and status/maintenance requests. Certain commands cannot be issued by VRM components, and the queue elements and CCBs are slightly different depending on the request from the virtual machine. The input requests for VRM components and virtual machines are all described in the following sections.

VRM Component Requests

From within the VRM, only I/O requests can be made to the SCSI device driver. The minidisk manager, coprocessor, and virtual memory manager use the following general purpose queue element to request the **read**, **write**, and **position** disk commands:

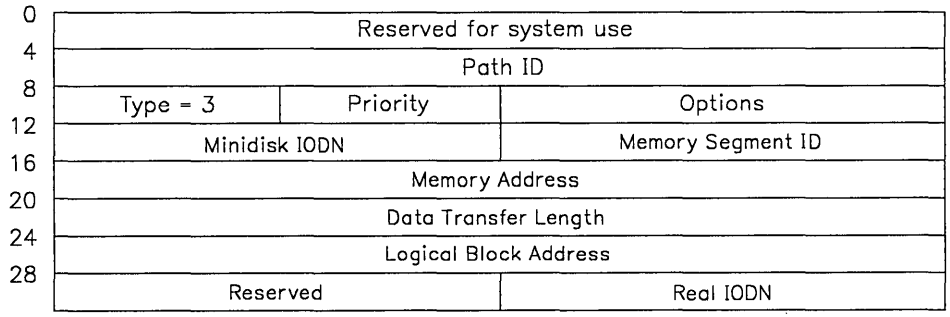


Figure 5-17. SCSI General Purpose Queue Element

The fields in the preceding figure are defined as follows:

Path ID: ID of the requesting component-to-SCSI device driver path

Type: 3 for general-purpose queue element

Priority: Not used

Options: Bits 16-26 of this field contain flags used to control the operation. Bits 27-31 contain the op code of the requested operation. Possible values for these fields include:

Bit 16 – set to one for interrupt on completion
Bit 17 – set to one for interrupt on error
Bit 18 – set to one if this is a synchronous operation
Bit 19 – always set to zero (no command extension)

Bit 20 – reserved
Bit 21 – set to one by **_mdmchk** if bad blocks are present
Bit 22 – set to one by **_mdmchk** for write verify
Bits 23-25 – reserved
Bit 26 – set to zero for a virtual address, one for a real address.

Bits 27-31 contain the op code for the request. Possible values include:

0 = Read

1 = Write

2 = Position

All other op codes are invalid for use with a general-purpose queue element.

Minidisk IODN:
IODN of the minidisk on which the operation is to be performed

Memory Segment ID:
ID of the segment that contains the data buffer

Memory Address:
Address of the buffer in the segment

Data Transfer Length:
Length of bytes of the data transfer

Logical Block Address:
Location on the minidisk where the operation is to be performed.

This is a value from zero to n, where zero is the first sector on cylinder 0 of the disk and n is the last sector on the last data cylinder. This value comes from the virtual machine as a logical block number on a minidisk. The minidisk manager converts this value into an offset from the start of the real disk. The SCSI device driver converts this to the corresponding cylinder, head, and sector values to access the data on the disk.

Real IODN: Indicates the IODN of the disk unit that contains the minidisk.

Virtual Machine Requests

Virtual machine requests to the SCSI device driver are initiated by **Start I/O SVCs**. One of the parameters to the **Start I/O SVC** is the address of a CCB. From the data contained in a CCB, the VRM builds a Start I/O queue element and sends it to the device driver. The CCB received from the virtual machine will have different fields depending on the requested operation. All the possible CCB fields for SCSI requests are shown in the following figure.

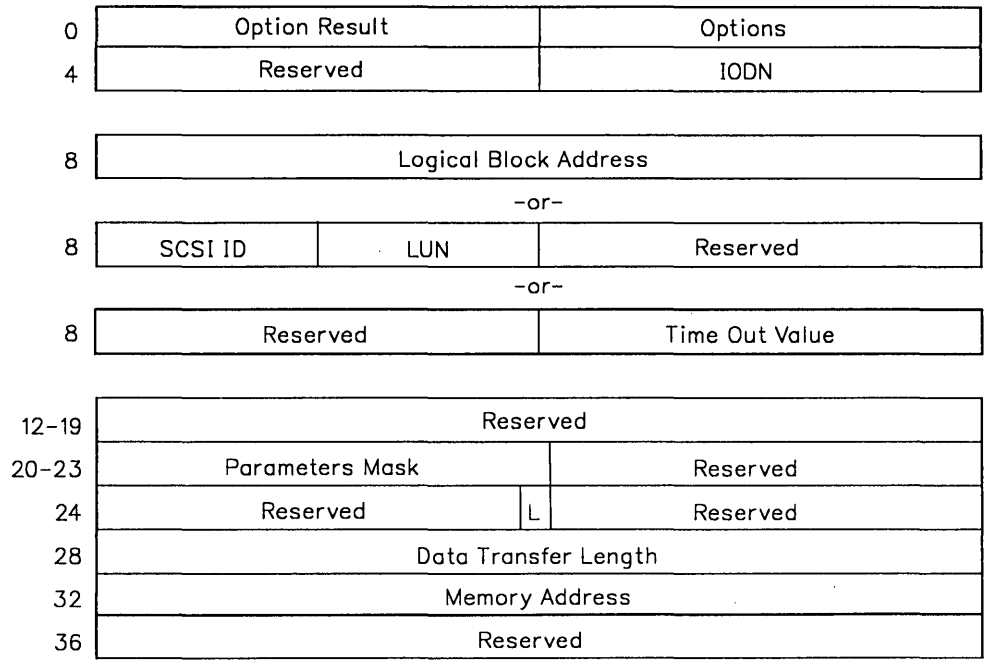


Figure 5-18. SCSI Command Control Block

The fields in the preceding figure are defined as follows:

Operation Result:
Copy of the PSB operation results field if synchronous operation is specified.

Options: Bits 16-26 of this field contain flags used to control the operation. These bits are defined with each operation. Bits 27-31 contain the op code of the requested operation. Possible values for these fields include:

0 = Read

1 = Write

2 = Position

6 = Read status

11 = Basic assurance test

14 = Change parameters

16 = Diagnostic ID information

17 = Format

18 = Device ID information

19 = Read capacity

20 = Reassign block

21 = Request sense

22 = Read server area

23 = Write server area

24 = Inquiry

25 = Reserved

26 = Wrap test

27 = Passthrough

28 = Mode sense

29 = Rewind

30 = Write file marks.

Each option is described in more detail in "Virtual Machine Command Description" on page 5-104.

IODN: Of the disk (or other SCSI device)

Bytes 8-11 can vary depending on the request. This word will have one of the following definitions:

Logical Block Address:
Indicates the location where the operation is to be performed.

This is a value from zero to n, where zero is the first sector on cylinder 0 of the disk and n is the last sector on the last data cylinder.

SCSI ID, LUN:
Some requests specify the SCSI ID and LUN of the disk from which information is required.

Time Out Value:
The Time Out Value indicates the number of 6-second timer ticks to allow before the device driver times out for the device with the specified IODN. This field is valid only for **Change Parameters** operations and only if bit 0 of the Parameters Mask (byte offset 20) is set to 1. If bit 0 of the Parameters Mask is set to 1 and the Time Out Value field contains 10, the device driver will time out after 60 (10 times 6 seconds per tick) seconds. This value remains in effect until the user sets a new value or the system is re-IPLed.

Parameters Mask:
This halfword indicates whether the Time Out Value is valid and specifies the data transfer type. When bit 0 is set to 1, the Time Out Value (byte offset 10-11) field is valid; when bit 0 is set to 0, the Time Out Value is ignored. Bits 2 and 3 together determine the transfer type in effect. These bits are defined as follows (in binary):

00	Continue running at current setting.
01	Transfer data in variable lengths.
10	Transfer data in block length.
11	Invalid (do not use).

If you are transferring variable lengths, your application must start and stop transfers on even-numbered addresses, and only multiples of four bytes can be transferred because the adapter has a 16-bit interface.

L: The link bit is not used with SCSI requests because chained command elements are not needed.

Data Transfer Length:
Length in bytes of the data buffer.

Memory Address:
Location of the data buffer.

The Start I/O queue element that is built from the data contained in the CCB is shown in Figure 5-19 on page 5-103.

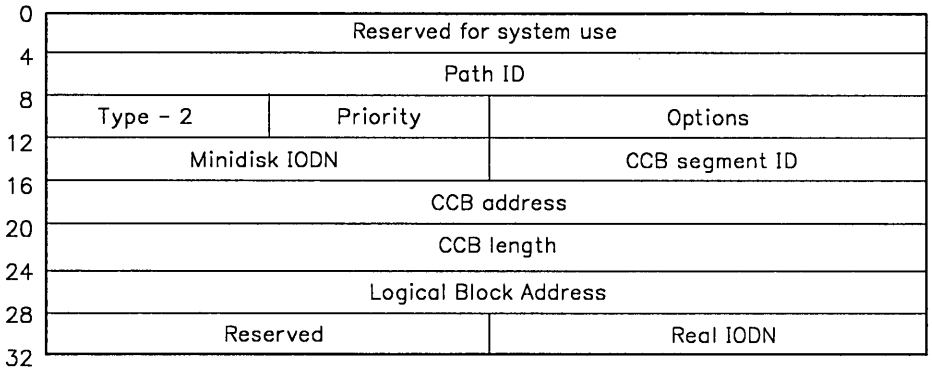


Figure 5-19. SCSI Start I/O Queue Element

The fields in the preceding figure are defined as follows:

Path ID: ID of the requesting component-to-SCSI device driver path

Type: 2 for Start I/O queue element.

Priority: Not used.

Options: This field is copied from the CCB.

Minidisk IODN:
IODN of the minidisk on which the operation is to be performed.

CCB Segment ID:
ID of the segment that contains the CCB.

CCB Address:
Address in the segment of the CCB.

CCB Length:
Length of bytes of the CCB.

Logical Block Address:
Location on the minidisk where the operation is to be performed.

This is a value from zero to n, where zero is the first sector on cylinder 0 of the disk and n is the last sector on the last data cylinder. This value comes from the virtual machine as a logical block number on a minidisk. The minidisk manager converts this value into an offset from the start of the real disk. The SCSI device driver converts this to the corresponding cylinder, head, and sector values to access the data on the disk.

Real IODN: Indicates the IODN (SCSI ID plus LUN) of the disk unit that contains the minidisk.

Virtual Machine Command Description

The following operation options to the SCSI device driver can be requested by a virtual machine. Each option is described with the CCB type (value of CCB bytes 8-11) and operation options flags defined. The virtual machine-to-SCSI device driver commands include:

- **Read** (opcode 0)

  This command is the standard disk **read**. Bytes 8-11 of the CCB contain a logical block address.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion

  Bit 17 = interrupt on error

  Bit 18 = synchronous operation

  Bit 19 = command extension

  Bit 20 = reserved for VRM

  Bit 21 = bad blocks present

  Bit 22 = reserved (must be 0)

  Bit 23 = reserved (must be 0)

  Bits 24-25 = ignored

  Bit 26 = reserved (must be 0).

- **Write** (opcode 1)

  This command is the standard disk **write**. Bytes 8-11 of the CCB contain a logical block address. Bits 16-26 of the operation options field are the same as for the **read** command described above, except that bit 22 is defined as write verify enabled. Write verify is supported only for 9332 DASD attachments.

- **Position** (opcode 2)

  This command is the standard disk **position**. Bytes 8-11 of the CCB contain a logical block address. Bits 16-26 of the operation options field are the same as for the **read** command described above.

- **Read status** (opcode 6)

  This command allows you to see the last command issued to the device and the return status from the device and adapter. It performs no operation on the SCSI bus, but merely copies data (16 bytes of command storage and 4 bytes of result storage) from the device driver's storage to the user's buffer. Bytes 8-11 of the CCB are reserved.

The operation options flags (bits 16-26) for this request are defined as follows:

Bit 16 = interrupt on completion

Bit 17 = interrupt on error

Bit 18 = synchronous operation

Bit 19 = command extension

Bit 20 = reserved for VRM

Bits 21-25 = ignored

Bit 26 = reserved (must be 0).

- **Basic assurance test** (opcode 11)

  This command tests the adapter's registers, hardware,and memory for proper function. This command stops any activity being done by the adapter, so this command should be issued only when the adapter is inactive. Bytes 8-11 of the CCB are reserved for this command.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion

  Bit 17 = interrupt on error

  Bit 18 = synchronous operation

  Bit 19 = command extension

  Bit 20 = reserved for VRM

  Bits 21-25 = ignored

  Bit 26 = reserved (must be 0).

- **Change parameters** (opcode 14)

  This command allows you to change certain selected parameters for SCSI devices. These parameters include the number of 6-second timer ticks allowed before the device driver times out, and the type of data transfer from the adapter to the device. This command uses the Time Out Value and Parameters Mask fields shown in Figure 5-18 on page 5-101.

- **Diagnostic ID information** (opcode 16)

  This command allows you to query information about a device without being attached to the device. Bytes 8 and 9 of the CCB contain the SCSI ID and the LUN, respectively, of the device being queried. The device driver first does an **inquiry** command to the specified device and places the first 36 bytes of the result of the inquiry into the caller's buffer. Next, the driver does a **request sense** command and also places the

results of that request at byte offset 36 of the caller's buffer. The data length field of the CCB should be set to 512.

The operation options flags (bits 16-26) for this request are defined as follows:

Bit 16 = interrupt on completion

Bit 17 = interrupt on error

Bit 18 = synchronous operation

Bit 19 = command extension

Bit 20 = reserved for VRM

Bits 21-25 = ignored

Bit 26 = reserved (must be 0).

- **Format** (opcode 17)

  The **format** command is used to physically reformat the individual disk units of a 9332 DASD Attachment. This command requires a command element. Each disk unit has a server area and a CE area on it, but these areas are not affected. Two levels of **format** are provided, depending on whether you want to preserve the disk unit's grown defect map. A bit mask in the operation options flags field (bit 22) indicates the level. For level 1 (bit 22 = 1), the format ignores the grown defect map of the disk unit and rewrites the entire file using only the permanent defect map. For level 2 (bit 22 = 0), the format uses both the grown defect and permanent defect maps. The user must provide a defect list, specifying the address and length of the defect list in a CCB command element. For the 9332, the defect list consists of four bytes of 0.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion
  Bit 17 = interrupt on error
  Bit 18 = synchronous operation
  Bit 19 = command extension
  Bit 20 = reserved for VRM
  Bit 21 = ignored
  Bit 22 = format level mask. When bit 22 = 1, a level 1 format is performed. When bit 22 = 0, a level 2 format is performed.
  Bits 23-25 = ignored
  Bit 26 = reserved (must be 0).

- **Device ID information** (opcode 18)

  This command allows you to determine information about a device attached to an adapter. The device driver uses the SCSI ID and LUN specified in the CCB to get information needed by the caller to define the device. CCB bytes 8 and 9 contain the SCSI ID and LUN, respectively, of the device. The device driver then does an **inquiry** of the specified device and uses DMA to place the first 36 bytes of the inquiry results

(these bytes include the device type as defined by the ANSI standard for SCSI devices) in the caller's buffer. Next, the driver does a **read capacity** command and adds the returned information to the caller's buffer. The operation options flags (bits 16-26) for this request are defined as follows:

Bit 16 = interrupt on completion

Bit 17 = interrupt on error

Bit 18 = synchronous operation

Bit 19 = command extension

Bit 20 = reserved for VRM

Bits 21-25 = ignored

Bit 26 = reserved (must be zero).

- **Read capacity** (opcode 19)

  This SCSI command allows you to determine the capacity of a device attached to the SCSI bus. Bytes 8-11 of the CCB are reserved. The data length field should be set to 512. The returned data includes four bytes for the address of the last logical block on the LUN (0x0005F80E for a 9332 DASD attachment) and four bytes for the block length of the last logical block on the LUN (0x200 for a 9332 DASD).

  Bit 21 of the operations options flags indicates whether the command is performed for the entire device or for a single track on the device.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion

  Bit 17 = interrupt on error

  Bit 18 = synchronous operation

  Bit 19 = command extension

  Bit 20 = reserved for VRM

  Bit 21 = Read capacity of device or track

  When bit 21 = 0, the **read capacity** is performed for the entire device.

  When bit 21 = 1, the **read capacity** is done for a specified track only. The specified track is determined by the track that contains the selected logical block address.

  Bits 22-25 = ignored

  Bit 26 = reserved (must be zero).

- **Reassign block** (opcode 20)

  This SCSI command allows you to reassign bad blocks on the 9332 DASD Attachment. This command causes data in the area surrounding the reassigned block to be shifted and sector IDs to be modified. The data contained in the block being reassigned will be altered by the target, so the initiator should read and save the data before issuing this

command. After successfully reassigning the block, the initiator can then write the data to the new location.

IBM recommends that you leave the 9332 DASD in auto reassign mode, allowing the VRM to handle bad-block management tasks.

The caller must provide the address of the buffer that contains the defective block list in the following format:

- Bytes 0-1 = reserved (set to zero)
- Byte 2 = defect length list MSB (0 for 9332 DASD)
- Byte 3 = defect length list LSB (4 for 9332 DASD)
- Byte 4 = defective logical block address MSB
- Byte 5 = defective logical block address
- Byte 6 = defective logical block address
- Byte 7 = defective logical block address LSB.

Bytes 8-11 of the CCB contain a logical block address.

The operation options flags (bits 16-26) for this request are defined as follows:

Bit 16 = interrupt on completion

Bit 17 = interrupt on error

Bit 18 = synchronous operation

Bit 19 = command extension

Bit 20 = reserved for VRM

Bits 21-25 = ignored

Bit 26 = reserved (must be zero).

- **Request sense** (opcode 21)

  This SCSI command allows you to discover if there are problems with a SCSI device. This command returns sense data from a prior check condition on an LUN. The virtual machine must supply a buffer in the CCB into which returned data is placed. The data transfer length should be specified as 512.

  This command is performed automatically by the device driver when a **read** or **write** command is done to the disk. To obtain the results of such a **request sense** operation, do a **read status** call.

  Bytes 8-11 of the CCB are reserved.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion

  Bit 17 = interrupt on error

Bit 18 = synchronous operation

Bit 19 = command extension

Bit 20 = reserved for VRM

Bits 21-25 = ignored

Bit 26 = reserved (must be zero).

- **Read server area** (opcode 22)

  This command allows you to read the server area of a 9332 DASD Attachment. The server area contains adapter microcode and other device-dependent data. This command allows you to determine EC levels and other data about the device. Bytes 8-11 of the CCB contain a logical block address in the range 0-227.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion

  Bit 17 = interrupt on error

  Bit 18 = synchronous operation

  Bit 19 = command extension

  Bit 20 = reserved for VRM

  Bits 21-22 = ignored

  Bit 23 = reserved (must be zero)

  Bit 24-25 = ignored

  Bit 26 = reserved (must be zero).

- **Write server area** (opcode 23)

  This command allows you to write to the server area of a 9332 DASD Attachment. The server area contains adapter microcode and other device-dependent data. This command allows you to download microcode to the server area and change EC levels. Bytes 8-11 of the CCB contain a logical block address in the range 0-227.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion

  Bit 17 = interrupt on error

  Bit 18 = synchronous operation

Bit 19 = command extension

Bit 20 = reserved for VRM

Bits 21-25 = ignored

Bit 26 = reserved (must be zero).

- **Inquiry** (opcode 24)

  This command allows you to find out information about a SCSI device. Returned information includes device type and data on the device vendor. Bytes 8-11 of the CCB are reserved.

  Data transfer length should be set to 512, but the first returned byte indicates the number of valid bytes in the transfer.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion

  Bit 17 = interrupt on error

  Bit 18 = synchronous operation

  Bit 19 = command extension

  Bit 20 = reserved for VRM

  Bits 21-25 = ignored

  Bit 26 = reserved (must be zero).

- **Wrap test** (opcode 26)

  This command causes the adapter to internally connect its outputs to its inputs to check the flow of data. All devices should be unplugged from the SCSI bus when this diagnostic command is issued. Bytes 8-11 of the CCB are reserved.

  Note that no other SCSI commands should be outstanding when you issue the **wrap test** command.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion

  Bit 17 = interrupt on error

  Bit 18 = synchronous operation

  Bit 19 = command extension

  Bit 20 = reserved for VRM

  Bits 21-25 = ignored

  Bit 26 = reserved (must be zero).

- **Passthrough** (opcode 27)

  This command allows you to pass a request directly to a device. SCSI commands of 6, 10, or 12 bytes are passed in the CCB. There is a 6K-byte limit on data transfers with **passthrough**. Note the extensive use of the operations options flags bits below.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion

  Bit 17 = interrupt on error

  Bit 18 = synchronous operation

  Bit 19 = command extension

  Bit 20 = reserved for VRM

  Bit 21 = 6-byte command

  When bit 21 = 1, a 6-byte command is contained in CCB bytes 8-13.

  Bit 22 = 10-byte command

  When bit 22 = 1, a 10-byte command is contained in CCB bytes 8-17.

  Bit 23 = 12-byte command

  When bit 23 = 1, a 12-byte command is contained in CCB bytes 8-19.

  Bit 24 = Data phase

  When bit 24 = 0, there is no data phase for this command. When bit 24 = 1, there is data for the command.

  Bit 25 = Data direction

  Bit 25 indicates the direction of the data when bit 24 indicates the command has a data phase (bit 24 = 1). When bit 25 = 0, the data is coming in from the device. When bit 25 = 1, data is going out to the device.

  Bit 26 = reserved (must be zero).

- **Mode sense** (opcode 28)

  This command allows a target device on a SCSI bus to report its medium, LUN, and parameters to the initiator. Bytes 8-11 of the CCB are reserved.

  Data transfer length should be set to 512, but the first returned byte indicates the number of valid bytes in the transfer.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion

  Bit 17 = interrupt on error

  Bit 18 = synchronous operation

  Bit 19 = command extension

  Bit 20 = reserved for VRM

  Bits 21-25 = ignored

  Bit 26 = reserved (must be zero).

- **Rewind** (opcode 29)

  This command causes the device driver to issue a rewind command to the tape device. The rewind is performed immediately, and status is returned upon completion of the rewind.

  The IODN specified in the CCB is the IODN of the tape drive. Bytes 8-11 of the CCB are reserved.

  The operation options flags (bits 16-26) for this request are defined as follows:

  Bit 16 = interrupt on completion

  Bit 17 = interrupt on error

  Bit 18 = synchronous operation

  Bit 19 = command extension

  Bit 20 = reserved for VRM

  Bits 21-25 = ignored

  Bit 26 = reserved (must be zero).

- **Write File Marks** (opcode 30)

  This command causes the specified number of filemarks to be written beginning at the current medium position on the tape.

  The IODN specified in the CCB is the IODN of the tape drive. Bytes 8-11 of the CCB indicate the number of filemarks to be written.

The operation options flags (bits 16-26) for this request are defined as follows:

Bit 16 = interrupt on completion

Bit 17 = interrupt on error

Bit 18 = synchronous operation

Bit 19 = command extension

Bit 20 = reserved for VRM

Bits 21-25 = ignored

Bit 26 = reserved (must be zero).

Output Values

The SCSI device driver uses an acknowledgment queue element to inform the requestor of the completion of the request. The format of this queue element is shown in the following figure.

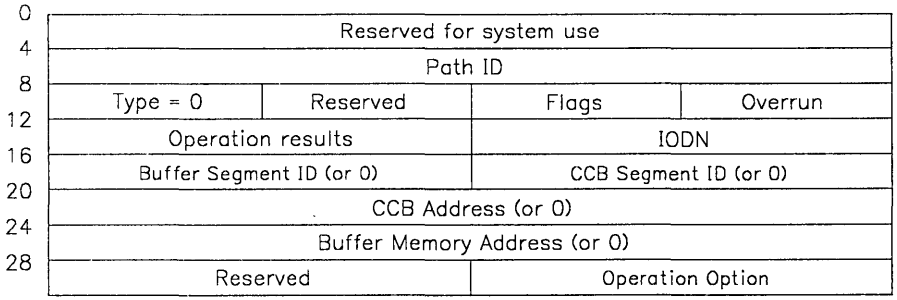


Figure 5-20. SCSI Acknowledgment Queue Element

The fields in the preceding figure are defined as follows:

Path Id: ID of the requestor-to-SCSI device driver path

Type: 0 for acknowledgment queue element

Flags: Set to 0x24 to indicate a solicited Start I/O interrupt

Overrun: Always set to zero

Operation Results:
Contains the completion status of the operation

IODN: Of the SCSI device that completed the operation

Buffer Segment ID:
For general-purpose requests, this field contains the segment ID specified in the input queue element. For Start I/O requests, this field contains zero.

CCB Segment ID:
For Start I/O requests, this field contains the CCB segment ID specified in the input queue element. For general-purpose requests, this field contains zero.

CCB Address: For Start I/O requests, this field contains the CCB address specified in the input queue element. For general-purpose requests, this field contains zero.

Buffer Memory Address:
For general-purpose requests, this field contains the memory address specified in the input queue element. For Start I/O requests, this field contains zero.

Operation Option:
For general-purpose requests, this field contains a copy of the operation options field of the input queue element. For Start I/O requests, this field contains zero.

If the device driver is unable to perform a requested operation or an error occurs during an operation, the interrupt handling routine returns the following values:

-02 = Time out
-03 = SCSI adapter busy, resubmit
-04 = No controller
-05 = Bus error from the adapter
-06 = No SCSI device
-07 = Check condition (request sense suggested)
-08 = 9332 DASD interface process failure
-16 = 9332 DASD file processor failure on LUN 0
-17 = 9332 DASD file processor failure on LUN 1
-32 = 9332 DASD servo failure for servo on LUN 0
-40 = Bad block
-64 = 9332 DASD servo failure for servo on LUN 1
-128 = Adapter failed
-192 = Other SCSI device in error
257 = Invalid or undefined opcode
258 = Invalid data length
259 = Invalid parameter(s).

The operation options flags (bits 16-26) for this request are defined as follows:

Bit 16 = interrupt on completion

Bit 17 = interrupt on error

Bit 18 = synchronous operation

Bit 19 = command extension

Bit 20 = reserved for VRM

Bits 21-25 = ignored

Bit 26 = reserved (must be zero).

Output Values

The SCSI device driver uses an acknowledgment queue element to inform the requestor of the completion of the request. The format of this queue element is shown in the following figure.

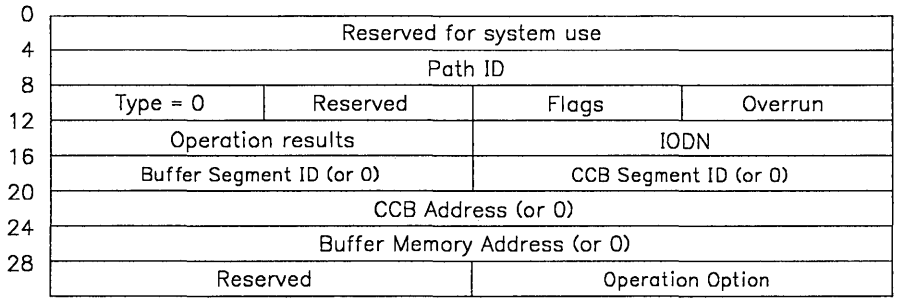


Figure 5-20. SCSI Acknowledgment Queue Element

The fields in the preceding figure are defined as follows:

Path Id: ID of the requestor-to-SCSI device driver path

Type: 0 for acknowledgment queue element

Flags: Set to 0x24 to indicate a solicited Start I/O interrupt

Overrun: Always set to zero

Operation Results:
Contains the completion status of the operation

IODN: Of the SCSI device that completed the operation

Buffer Segment ID:
For general-purpose requests, this field contains the segment ID specified in the input queue element. For Start I/O requests, this field contains zero.

CCB Segment ID:
For Start I/O requests, this field contains the CCB segment ID specified in the input queue element. For general-purpose requests, this field contains zero.

CCB Address: For Start I/O requests, this field contains the CCB address specified in the input queue element. For general-purpose requests, this field contains zero.

Buffer Memory Address:
For general-purpose requests, this field contains the memory address specified in the input queue element. For Start I/O requests, this field contains zero.

Operation Option:
For general-purpose requests, this field contains a copy of the operation options field of the input queue element. For Start I/O requests, this field contains zero.

If the device driver is unable to perform a requested operation or an error occurs during an operation, the interrupt handling routine returns the following values:

-02 = Time out
-03 = SCSI adapter busy, resubmit
-04 = No controller
-05 = Bus error from the adapter
-06 = No SCSI device
-07 = Check condition (request sense suggested)
-08 = 9332 DASD interface process failure
-16 = 9332 DASD file processor failure on LUN 0
-17 = 9332 DASD file processor failure on LUN 1
-32 = 9332 DASD servo failure for servo on LUN 0
-64 = 9332 DASD servo failure for servo on LUN 1
-128 = Adapter failed
-192 = Other SCSI device in error
257 = Invalid or undefined opcode
258 = Invalid data length
259 = Invalid parameter(s).

Streaming Tape Device Driver

This device driver supports the IBM RT PC Streaming Tape Drive Adapter, which is used to control a single IBM RT PC Streaming Tape Drive. This is a nonshared device driver. Therefore, only one virtual machine can be attached to this device driver at a time.

The DDS for the streaming tape device driver is created by an lpost that passes the DDS to the load list processor. For information on the load list processor, see "VRM IPL and Configuration" on page 1-4. The streaming tape device driver is unique in that it uses a DDS that does not contain a device characteristics section. The streaming tape device driver is defined on the following pages.

Define Device Header

This section of the DDS is common to all device drivers. The information is defined as follows:

IODN: 11

IOCN: 0x0270

Define Option:
0x0001

Device Type: 0x0001

Device Name: This field is ignored by the VRM. It can be used to store a name for the device that uses this device driver.

Offset to Hardware Characteristics:
28 bytes

Offset to Device Characteristics:
0 bytes because this section is not used for this device driver.

Offset to Error Log:
72 bytes

Hardware Characteristics

This section of the DDS is common to all device drivers. The information is defined as follows:

Length: 11 words

Internal device type:
0xA1xx8500

(where xx indicates the slot occupied by the adapter).

Base I/O port address:
0x000001E8

Number of I/O port addresses:
7

Bus memory start address:
Not used by this device driver.

Bus memory end address:
Not used by this device driver.

DMA type: Not used by this device driver.

Interrupt type:
0xC000010C

- Interrupts enabled
- Interrupts shareable
- Class = 1
- Interrupt level = 12.

Error Log

The error log section for this device driver is structured as shown in Figure 5-21:

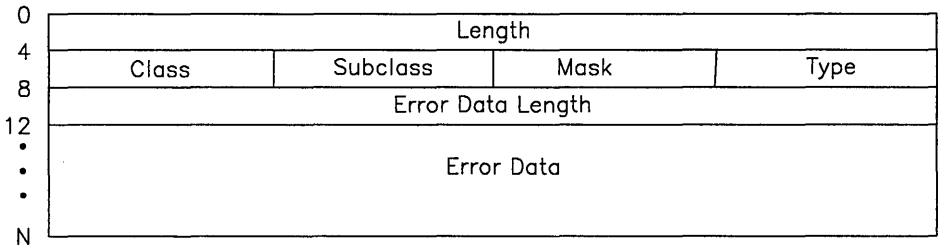


Figure 5-21. Error Log Structure

The various fields of the error log section of the DDS are defined as follows:

Length: 7 words

Class: 0x01 to indicate hardware error.

Subclass: 0x04

Mask: 0x00

Type: Can be one of two values:

- 0x40 to indicate a temporary error
- 0x80 to indicate a permanent error.

Length of error data:
5 words

Error data: Errors can be detected for three different components: the device driver, the IBM RT PC Streaming Tape Drive Adapter, and the IBM RT PC Streaming Tape Drive. Only the IBM RT PC Streaming Tape Drive performs retries upon getting an initial error. The device driver and the IBM RT PC Streaming Tape Drive Adapter do not have enough information or control to perform retries. Therefore, all errors are reported back to the virtual machine except those that are temporary drive errors.

The only information kept about temporary drive errors is the number of errors encountered during the entire streaming task. These are called soft errors and are kept in the soft count field of the error data.

The error data field can be structured in three different ways depending on the type of error that occurred. The first byte of error data indicates the type of error as follows:

- 0x00 — Adapter error
- 0x10 — Drive error
- 0x20 — Tape drive self-test error.

The values of the remaining bytes depend on the type of error detected. The following sections describe the error data field structure for each of the three types of errors.

Error Data for Adapter Errors

For IBM RT PC Streaming Tape Drive Adapter errors, the error data is structured as shown in Figure 5-22:

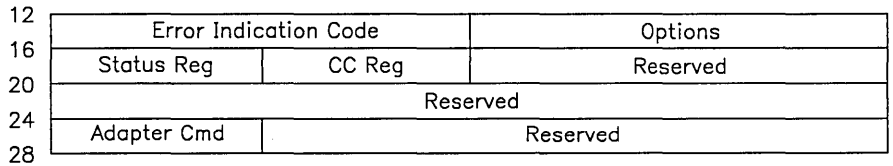


Figure 5-22. Error Data for Adapter Errors

The various fields of the Error Data are defined as follows:

Error indication code:

The first byte indicates that this is an adapter error. It is always set to 0x00. The second byte indicates the specific error as follows:

- 0x00 — Drive not ready
- 0x01 — Drive ready error
- 0x02 — Adapter parity check
- 0x03 — Adapter failure
- 0x04 — Transfer not complete
- 0x05 — Invalid adapter command
- 0x06 — Command or data sequence error
- 0x07 — Adapter ROM checksum error
- 0x08 — Adapter RAM error
- 0x09 — Data transfer logic error
- 0x0A — Drive not reset error
- 0x0B — Data bus error
- 0x0C — Unexpected adapter power-on reset
- 0x0D — Unknown adapter error
- 0x0E — Read status failure.

Options: The operation option and device option sent to the device driver.

Status register:
Last status from the IBM RT PC Streaming Tape Drive Adapter status port. See "Status Register Values" on page 5-123 for the complete definition.

Completion code register:
Last completion code from the IBM RT PC Streaming Tape Drive Adapter. See "Completion Code Register Values" on page 5-124 for the complete definition.

Adapter command:
Last command given to the IBM RT PC Streaming Tape Drive Adapter. See "Adapter Command Values" on page 5-124 for the complete definition.

Error Data for Drive Errors

For IBM RT PC Streaming Tape Drive errors, the error data is structured as shown in Figure 5-23:

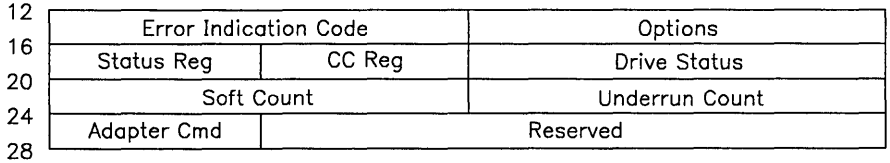


Figure 5-23. Error Data for Drive Errors

The various fields of the error data are defined as follows:

Error indication code:
The first byte indicates that this is a drive error. It is always set to 0x10. The second byte indicates the specific error as follows:

- 0x00 — Temporary read error
- 0x01 — Temporary write error
- 0x02 — Device fault
- 0x03 — Write abandoned
- 0x04 — Read abandoned
- 0x05 — Read error: bad block transfer
- 0x06 — Read Error: filler block transfer
- 0x07 — Read Error: no data detected
- 0x08 — Invalid drive command
- 0x09 — BOT not detected
- 0x0A — Unexpected drive power-on reset
- 0x0B — Unknown drive error
- 0x0C — Drive parity error.

Options: The operation option and device option sent to the device driver.

Status register:
Last status from the IBM RT PC Streaming Tape Drive Adapter status port. See "Status Register Values" on page 5-123 for the complete definition.

Completion code register:
Last completion code from the IBM RT PC Streaming Tape Drive Adapter. See "Completion Code Register Values" on page 5-124 for the complete definition.

Drive status: Last status received from the IBM RT PC Streaming Tape Drive. The drive status field is two bytes long and uses each bit to represent a different condition. If a bit is set to one, that condition is in effect.

The possible values for the first byte are as follows:

- Bit 0 — One or more bits set in this byte
- Bit 1 — No cartridge
- Bit 2 — Device fault
- Bit 3 — Write protected cartridge
- Bit 4 — End of media
- Bit 5 — Unrecoverable data error
- Bit 6 — Bad block not located
- Bit 7 — File mark detected.

The possible values for the second byte are as follows:

- Bit 0 — One or more bits set in this byte
- Bit 1 — Illegal command
- Bit 2 — No data detected
- Bit 3 — Marginal block detected
- Bit 4 — Beginning of media
- Bit 5 — Parity error
- Bit 6 — End of recorded media
- Bit 7 — Power-on reset occurred.

Soft count: The number of temporary errors that occurred.

Underrun count:
The number of underruns that occurred.

Adapter command:
Last command given to the IBM RT PC Streaming Tape Drive Adapter. See "Adapter Command Values" on page 5-124 for the complete definition.

Error Data for Drive Self-Test Commands

If a self-test command to the IBM RT PC Streaming Tape Drive fails, the error data section is structured as shown in Figure 5-24:

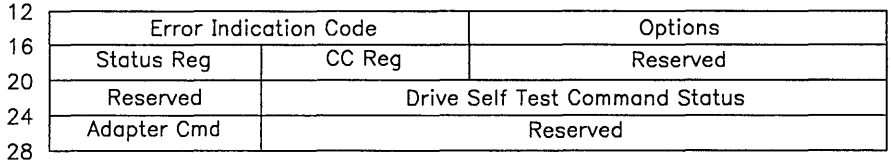


Figure 5-24. Error Data for Drive Self-Test Errors

The various fields of the error data are defined as follows:

Error indication code:
The first byte indicates that this is a self-test command error. The first byte is set to 0x20. The second byte is set to 0x00.

Options: The operation option and device option sent to the device driver.

Status register:
Last status from the IBM RT PC Streaming Tape Drive Adapter status port. See "Status Register Values" on page 5-123 for the complete definition.

Completion code register:
Last completion code from the IBM RT PC Streaming Tape Drive Adapter. See "Completion Code Register Values" on page 5-124 for the complete definition.

Drive self-test command status:
The last status returned from the drive following one of the drive self-test commands. The status consists of three bytes whose values depend on the test performed. The combination of the values in each byte indicates the specific error detected. There are different errors for each of the three tests.

The first byte can contain the following values:

- 0x00 — Self-test not performed
- 0x22 — Write/Read error
- 0x23 — Sensor or speed error
- 0x24 — Memory checksum error.

The second byte can contain the following values:

- 0x01 — Speed test failed
- 0x02 — Sensor test failed
- 0x03 — Write test failed

- 0x04 — Read test failed
- 0x05 — Erase test failed
- 0x06 — Write test with single data block failed
- 0x07 — CRC test failed
- 0x08 — Write test failed
- 0x09 — Read test failed.

The third byte can contain the following values:

- 0x01 — Sensor or speed error
- 0x02 — Head position error (if second byte is 0x07)
- 0x03 — Write timeout (if second byte is 0x03 or 0x06)
- 0x03 — Read timeout (if second byte is 0x04)
- 0x03 — Unable to erase (if second byte is 0x05)
- 0x03 — CRC failed (if second byte is 0x07)
- 0x04 — Unable to read
- 0x05 — Unable to read file mark
- 0x06 — Gap detect failure.

For drive self-test 1, only the first byte is used. The other two bytes are reserved. For drive self-test 2, only the first two bytes are used if a speed error or sensor error is detected. The third byte is reserved. In all other cases, all three bytes contain valid status information.

Adapter command:
Last command given to the IBM RT PC Streaming Tape Drive Adapter. See "Adapter Command Values" on page 5-124 for the complete definition.

Status Register Values

The status register field uses each bit to represent a different condition. If the bit is set to one, that condition is in effect. The possible values for the status register field are as follows:

- Bit 0 — Adapter busy
- Bit 1 — Drive ready
- Bit 2 — Online
- Bit 3 — Drive exception
- Bit 4 — Interrupt
- Bit 5 — Not used
- Bit 6 — Output buffer full
- Bit 7 — Input buffer full.

Completion Code Register Values

The possible values for the completion code register field are as follows:

- 0x00 — Good completion
- 0x01 — Load block count
- 0x02 — Awaiting power-on reset handshake
- 0x10 — Status available
- 0x30 — Read buffer full
- 0x31 — Read buffer full, last block
- 0x40 — Write buffer empty
- 0x55 — Power-on reset/Reset successful
- 0x80 — Exception status
- 0x81 — Drive not ready
- 0x82 — Ready error
- 0x83 — Drive parity check
- 0x84 — Direction error
- 0x85 — Data transfer did not complete
- 0x90 — Command reject, exception outstanding
- 0x91 — Invalid adapter command
- 0x92 — Command/Data sequence error
- 0xA0 — ROM sumcheck
- 0xA1 — RAM error
- 0xA2 — Data transfer logic failure
- 0xA3 — Drive not reset
- 0xA4 — Data bus error.

Adapter Command Values

The possible values for the adapter command field are as follows:

- 0x00 — Reset
- 0x02 — Rewind
- 0x03 — Erase
- 0x04 — Retension
- 0x05 — Read status
- 0x06 — Write
- 0x07 — Write file mark
- 0x08 — Read
- 0x09 — Read file mark
- 0x0C — Skip
- 0x0D — Terminate
- 0x0E — Drive self-test 1
- 0x0F — Drive self-test 2
- 0x10 — Drive self-test 3
- 0xAA — Power-on reset/Reset handshake.

CCB Input Values

This device driver accepts CCBs as shown in Figure 2-5 on page 2-12. The CCB consists of the command header and zero or more command elements, depending on the device option used.

This device driver uses the following definitions for the bits in the options field of the CCB. Note that this device driver uses eleven bits for the device option whereas other device drivers use some of the option bits for flags and only five bits for the actual device option.

- Bit 0 through 4 — Operation Option
- Bit 5 through 15 — Device Option.

A CCB header for this device driver has an additional field located at byte 12 in the CCB header. This field is the Block Count field and is used for the Write File Mark, Read File Mark, and Skip device options. It contains the number of file marks or blocks that the device option should operate upon.

Note that certain options are valid only in a specific mode. This device driver supports three mode types. They are:

- Read
- Write
- Maintenance

The Reset Device (5) and Terminate (14) options are valid in any mode. When either of these options is executed, the drive is placed in Maintenance mode.

The following options are valid in Read or Maintenance mode. If these options are issued from Maintenance mode, the drive is put in Read mode.

- Read (0)
- Read File Mark (11)
- Skip (13)

The following options are valid in Write or Maintenance mode. From Read mode, the following options are valid if the last option processed was a Read File Mark or if a 'No Data Detected' error occurred. If these options are issued from Maintenance mode, or after a Read File Mark or 'No Data Detected' error from Read mode, the drive is put in Write mode.

- Write (1)
- Write File Mark (10)

The following device options are valid only from Maintenance mode.

- Rewind (6)
- Erase (7)
- Retension (8)
- Drive Self Test 1 (15)

- Drive Self Test 2 (16)
- Drive Self Test 3 (17)
- Write Adapter Buffer Diagnostic (18)
- Read Adapter Buffer Diagnostic (19)

The device options are defined below. Device options that require a command element contain information about the command element in the definition. If there is no information about a command element, you can assume that the device option does not use a command element.

0 = Read

This device option reads data from the tape. There must be at least one command element with each Read device option. The command elements define the system data buffers used to receive the data. Each buffer element must begin on a word boundary and be a multiple of words in length. The total amount of buffer space defined by all the command elements must be a multiple of data blocks. A data block is 512 bytes.

This option is valid only if the tape drive is in Read or Maintenance mode.

1 = Write

This device option writes data to the tape. There must be at least one command element with each Write device option. The command elements define the system data buffers that contain the data to be sent to the IBM RT PC Streaming Tape Drive. Each buffer element must begin on a word boundary and be a multiple of words in length. The total amount of buffer space defined by all the command elements must be a multiple of data blocks.

This option is valid if the tape drive is in Write or Maintenance mode. It is also valid if the last option processed in Read mode was a Read File Mark or if a 'No Data Detected' error occurred.

2 = Position

This device option is not used by this device driver.

3 = Format

This device option is not used by this device driver.

4 = Change characteristics

This device option is not used by this device driver.

5 = Reset device

This device option is valid at any time and clears error situations.

6 = Rewind

This device option is valid only in Maintenance mode and rewinds to the beginning of the tape.

7 = Erase

This device option erases the entire tape, then rewinds to the beginning of the tape. You do not need to use a retension device option after using an erase device option because the erase device option performs the same type of function.

This option is valid only in Maintenance mode.

8 = Retension

This device option is used to exercise the tape cartridge. This option rewinds the tape, then winds to the end of the tape, then rewinds to the beginning of the tape. Use this device option if you have excessive read errors during a restore or if the tape has been exposed to environmental extremes.

This option is valid only in Maintenance mode.

9 = Read Status

This device option obtains the latest adapter and drive status information. Valid status is returned only after issuance of a Reset Device, Rewind, Erase, Retension, Terminate, or Drive Self Test 1-3 command. This device option requires a command element defining a system buffer to receive the 16 bytes of status information. The bytes are defined as follows:

- Bytes 0 and 1 — Drive status
- Bytes 2 and 3 — Soft error count
- Bytes 4 and 5 — Underrun count
- Byte 6 — Command received by the adapter
- Byte 7 — Adapter block count
- Byte 8 — Status register
- Byte 9 — Adapter completion code
- Byte 10 — Last command given to adapter
- Byte 11 to 15 — Reserved.

10 = Write file mark

This device option writes one or more file marks on the tape. This option also uses the block count field of the CCB Header. The block count field indicates the number of file marks to be written. The value can be from 1 to 255.

This device option is valid only in Write or Maintenance mode. It is also valid if the last option processed was a Read File Mark or if a 'No data detected' error occurred.

11 = Read File Mark

This device option is valid only in Read or Maintenance mode and reads one or more file marks on the tape. This option also uses the block count field of the CCB header. The block count field indicates the number of file marks to be read. The value can be from 1 to 255.

12 = This option is not used by this device driver.

13 = Skip

This device option is valid only in Read or Maintenance mode and skips one or more blocks on the tape. This option also uses the block count field of the CCB Header. The block count field indicates the number of blocks to skip. The value can be from 1 to the number of blocks remaining on the tape.

14 = Terminate

This option is valid in any mode. It ends the streaming task and rewinds to the beginning of the tape. If the tape was being written to, a file mark is placed on the tape before it is rewound.

15 = Drive self-test 1

This device option is a diagnostic command used to perform a checksum calculation on the program memory in the IBM RT PC Streaming Tape Drive.

This option is valid only in Maintenance mode.

16 = Drive self-test 2

This device option is a diagnostic command used to check the capstan speed and verify the operation of the BOT, EOT, and SAFE sensors. It writes a pattern on the tape and verifies it, verifies that the erase function is working, verifies that the CRC check function is correct, and checks that the tape head can be positioned properly.

This option is valid only in Maintenance mode.

17 = Drive self-test 3

This device option is a diagnostic command used to check the write-protect mechanism. You must use an erased, write-protected tape cartridge for this test.

This option is valid only in Maintenance mode.

18 = Write adapter buffer diagnostic

This device option is a diagnostic command used to write a 512-byte data pattern to the IBM RT PC Streaming Tape Drive Adapter buffer. A command element is required to define the 512-byte buffer containing the data pattern.

This option is valid only in Maintenance mode.

19 = Read Adapter Buffer Diagnostic

This device option is a diagnostic command used to read 512 bytes of data from the IBM RT PC Streaming Tape Drive Adapter buffer. A command element is required to define the 512-byte buffer that will receive the data.

This option is valid only in Maintenance mode.

20 through 31 = These device options are not used by this device driver.

Output Values

This section defines the output values that may appear in GPR2 or in a PSB. The PSB fields that contain I/O information or detected error information are the status flags field and the operation results field. Figure 2-6 on page 2-12 shows the location of these fields in the PSB.

The status flags field is a single byte and is defined as 0x24. This indicates that this is a solicited I/O interrupt.

The 16-bit operation results field returns a value indicating the completion status of the operation. The possible errors that can be returned are listed below:

00 = Successful completion

-02 = Invalid device option

The CCB contained an invalid device option.

-03 = Undefined device option

The CCB contained an undefined device option.

-04 = Invalid parameters

The CCB contained an invalid command parameter.

-05 = Streaming error

An invalid device option was sent to the device driver while performing a streaming read or write operation.

-10 = Drive not ready error

The adapter timed out while waiting for the drive to come to the ready state.

-11 = Drive ready error

The adapter timed out while waiting for the drive to come to the not ready state.

-12 = Adapter parity check

An error was detected in data being read from the drive.

-13 = Adapter failure

An error occurred while transferring data to or from the drive.

-14 = Transfer not complete

Data was not completely transferred between the adapter and the drive within the allowed time interval.

-15 = Invalid adapter command

The adapter received an invalid adapter command.

-16 = Command/Data sequence error

The adapter received a command when expecting data or it received data when expecting a command.

-17 = Adapter ROM checksum error

An adapter ROM checksum failure occurred.

-18 = Adapter RAM error

An adapter RAM failure occurred.

-19 = Data transfer logic error

A data transfer logic error occurred.

-20 = Drive not reset

The drive did not reset. This could occur if the drive is turned off.

-21 = Data bus error

A data bus error occurred.

-22 = Unexpected adapter POR or RESET

The adapter indicated a POR or RESET occurred even though it should not have occurred at this time.

-23 = Unknown adapter error

The adapter returned an undefined error code.

-24 = Read status failure

The adapter status could not be read.

-40 = Write protect

The tape cartridge is write-protected in this area.

-41 = No cartridge

The tape drive does not contain a tape cartridge.

-42 = End of media

The cartridge is positioned at the end of the tape.

-43 = File mark read

The device driver encountered a file mark block during a Read or Skip command. The operation can continue.

-44 = Device fault

Indicates one of the following conditions:

- The cartridge has stalled.
- The tape has not moved.
- The tape has broken.
- A write sequence error has occurred.

-45 = Write abandoned

Indicates one of the following conditions occurred during a Write or Write File Mark command:

- The same block was rewritten 16 or more times.
- An unrecoverable reposition error occurred.

The operation cannot continue.

-46 = Read abandoned

An unrecoverable reposition error occurred during a Read command. The operation cannot continue.

-47 = Read error, bad block transfer

The same block was read 16 times and always encountered a CRC error. The last block transferred contains data from the bad data block. The operation can continue.

-48 = Read error, filler block transfer

The same block was read 16 times and always encountered a CRC error. The last block transferred contains filler data. The operation can continue.

-49 = Read error, no data detected

The tape did not contain any recorded data. The operation cannot continue.

-50 = Invalid drive command

The drive received an invalid drive command.

-51 = BOT not detected

The device driver did not detect the beginning of the tape after a Rewind, Retension, Erase, or Terminate command.

-52 = Unexpected drive POR or RESET

The drive indicated a POR or RESET occurred even though it should not have occurred at this time.

-53 = Unknown drive error

The drive returned an undefined error code.

-54 = Drive parity error

The drive detected a parity error while writing to the tape.

-60 = Drive self-test failure

A drive failure occurred during a drive self-test command.

Glossary

address. A name, label, or number identifying a location in storage, a device in a system or network, or any other data source.

allocate. To assign a resource, such as a disk file or a diskette file, to perform a specific task.

All Points Addressable (APA) display. A display that allows each pel to be individually addressed. An APA display allows for images to be displayed that are not made up of images predefined in character boxes. Contrast with *character display.*

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

American National Standards Institute (ANSI). An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

ANSI. See *American National Standards Institute.*

a.out. An output file produced by default for certain instructions. By default, this file is executable and contains information for the symbolic debugger.

application. (1) A particular task, such as inventory control or accounts receivable. (2) A program or group of programs that apply to a particular business area, such as the Inventory Control or the Accounts Receivable application.

argument. Numbers, letters, or words that expand or change the way commands work.

array. An arrangement of elements in one or more dimensions.

ASCII. See *American National Standard Code for Information Interchange.*

assembler language. A symbolic programming language in which the set of instructions includes the instructions of the machine and whose data structures correspond directly to the storage and registers of the machine.

asynchronous transmission. In data communications, a method of transmission in which the bits included in a character or block of characters occur during a specific time interval. However, the start of each character or block of characters can occur at any time during this interval. Contrast with *synchronous transmission.*

attribute. A characteristic. For example, the attribute for a displayed field could be blinking.

bad block. A portion of a disk that can never be used reliably.

base register. A general purpose register that the programmer chooses to contain a base address. See also *index register.*

basic addressable unit (BAU). The smallest piece of storage that can be addressed.

BAU. See *basic addressable unit.*

binary. (1) Pertaining to a system of numbers to the base two; the binary digits are 0 and 1. (2) Involving a choice of two conditions, such as on-off or yes-no.

bit. Either of the binary digits 0 or 1 used in computers to store information. See also *byte.*

block. (1) A group of records that is recorded or processed as a unit. Same as *physical record.* (2) In data communications, a group of records that is recorded, processed, or sent as a unit. (3) A block is 512 bytes long.

boundary alignment. The position in main storage of a fixed-length field (such as halfword or doubleword) on an integral boundary for that unit of information. For example, a word boundary is a storage address evenly divisible by four.

bps. Bits per second.

branch. In a computer program an instruction that selects one of two or more alternative sets of instructions. A conditional branch occurs only when a specified condition is met.

breakpoint. A place in a computer program, usually specified by an instruction, where execution may be interrupted by external intervention or by a monitor program.

buffer. (1) A temporary storage unit, especially one that accepts information at one rate and delivers it at another rate. (2) An area of storage, temporarily reserved for performing input or output, into which data is read, or from which data is written.

bus. One or more conductors used for transmitting signals or power.

byte. The amount of storage required to represent one character; a byte is 8 bits.

call. (1) To activate a program or procedure at its entry point. Compare with *load.* (2) In data communications, the action necessary in making a connection between two stations on a switched line. by plugging it into a slot in the system unit.

channel. A path along which data passes. Also a device connecting the processor to input and output devices.

character. A letter, digit, or other symbol.

character display. A display that uses a character generator to display predefined character boxes of images (characters) on the screen. This kind of display can not address the screen any less than one character box at a time. Contrast with *All Points Addressable display.*

character set. A group of characters used for a specific reason; for example, the set of characters a printer can print or a keyboard can support.

character string. A sequence of consecutive characters.

C language. A general-purpose programming language that is the primary language of the AIX Operating System.

color display. A display device capable of displaying more than two colors and the shades produced via the two colors, as opposed to a monochrome display.

command line. The area to the right of a prompt for entering commands or program names.

compile. (1) To translate a program written in a high-level programming language into a machine language program. (2) The computer actions required to transform a source file into an executable object file.

compiler. A program that translates instructions written in a high-level programming language into machine language.

concatenate. (1) To link together. (2) To join two character strings.

configuration. The group of machines, devices, and programs that make up a computer system. See also *system customization.*

constant. A data item with a value that does not change. Contrast with *variable.*

control block. A storage area used by a program to hold control information.

control character. A non-printing character that performs formatting functions in a text file.

control program. Part of the AIX Operating System system that determines the order in which basic functions should be performed.

counter. A register or storage location used to accumulate the number of occurrences of an event.

cursor. (1) A movable symbol (such as an underline) on a display, used to indicate to the operator where the next typed character will be placed or where the next action will be directed. In Usability Services, the cursor is called a text cursor. (2) A marker that indicates the current data access location within a file. See also *pointing cursor.*

customize. To describe (to the system) the devices, programs, users, and user defaults for a particular data processing system.

cylinder. All fixed disk or diskette tracks that can be read or written without moving the disk drive or diskette drive read/write mechanism.

daemon. See *daemon process.*

daemon process. A process begun by the root or the root shell that can be stopped only by the root. Daemon processes generally provide services that must be available at all times such as sending data to a printer.

data stream. All information (data and control information) transmitted over a data link.

deadlock. An error condition in which processing cannot continue due to unresolved contention for the use of a resource.

debug. (1) To detect, locate, and correct mistakes in a program. (2) To find the cause of problems detected in software.

debugger. A device used to detect, trace, and eliminate mistakes in computer programs or software.

default. A value that is used when no alternative is specified by the operator.

default value. A value stored in the system that is used when no other value is specified.

device driver. A program that operates a specific device, such as a printer, disk drive, or display.

device manager. Collection of routines that act as an intermediary between device drivers and virtual machines for complex interfaces. For example, supervisor calls from a virtual machine are examined by a device manager and are routed to the appropriate subordinate device drivers.

device name. A name reserved by the system that refers to a specific device.

direct memory access (DMA) device. A component that can read or write to system storage directly, without processor intervention. Two device types are identified: 1) an alternate controller resides on a hardware adapter and 2) a system DMA controller resides on the system planar board. DMA capability permits simultaneous use of input/output devices and the processor.

directory. A type of file containing the names and controlling information for other files or other directories.

disable. To make nonfunctional.

displacement. A positive or negative number that can be added to the contents of a base register to calculate an effective address.

display device. An output unit that gives a visual representation of data.

DMA. See *direct memory access.*

dump. (1) To copy the contents of all or part of storage, usually to an output device. (2) Data that has been dumped.

EBCDIC. See *extended binary-coded decimal interchange code.*

ECB. See *event control bit.*

effective address. A real storage address that is computed at runtime. The effective address consists of contents of a base register, plus a displacement, plus the contents of an index register if one is present.

emulation. Imitation; for example, when one computer imitates the characteristics of another computer.

enable. To make functional.

entry point. An address or label of the first instruction performed upon entering a computer program, a routine, or a subroutine. A program may have several different entry points, each corresponding to a different function or purpose.

escape character. A character that changes the meaning of the characters that follow. For example the backslash character, used to indicate to the shell that the next character is not intended to have the special meaning normally assigned to it by the shell.

event. The enqueueing of an element.

event control bit (ECB). A bit assigned to each queue to signal the arrival or departure of an element.

exception handler. A set of routines used to detect deadlock conditions or to process abnormal condition processing. This allows the normal execution of processes to be interrupted and resumed.

expression. A representation of a value. For example, variables and constants appearing alone or in combination with operators.

extended binary-coded decimal interchange code (EBCDIC). A set of 256 eight-bit characters.

external symbol. A symbol that is defined in a file other than the file in which the symbol occurs. An ordinary symbol that represents an external reference.

file. A collection of related data that is stored and retrieved by an assigned name.

file system. The collection of files and file management structures on a physical or logical mass storage device, such as a diskette or minidisk.

first level interrupt handler (FLIH). A routine that receives control of the system as a result of a hardware interrupt. One FLIH is assigned to each of the six interrupt levels.

flag. A modifier that appears on a command line with the command name that defines the action of the command. Flags in the AIX Operating System almost always are preceded by a dash.

FLIH. See *first level interrupt handler.*

font. A family or assortment of characters of a given size and style.

format. (1) A defined arrangement of such things as characters, fields, and lines, usually used for displays, printouts, or files. (2) The pattern which determines how data is recorded.

function. A synonym for procedure. The C language treats a function as a data type that contains executable code and returns a single value to the calling function.

gather. For input/output operations, reading data from noncontiguous memory l locations to write to a device. See *scatter* (antonym).

general purpose register (GPR). An explicitly addressable register that can be used for a variety of purposes (for example, as an accumulator or an index register). Π has sixteen 32-bit GPRs. See *register.*

generation. For some remote systems, the translation of configuration information into machine language.

global variable. A symbol defined in one program module, but used in other independently assembled program modules.

GPR. See *general purpose register.*

granularity. The extent to which a larger entity is subdivided. For example, a yard broken into inches has finer granularity than a yard broken into feet.

graphic character. A character that can be displayed or printed.

header record. A record at the beginning of a file that details the sizes, locations, and other information that follows in the file.

hex. See *hexadecimal.*

hexadecimal. Pertaining to a system of numbers to the base sixteen; hexadecimal digits range from 0 (zero) through 9 (nine) and A (ten) through F (fifteen).

high-order. Most significant; leftmost. For example, bit 0 in a register.

history file. (1) A file that exists for each licensed program product supplied by IBM that documents the version, release, and level of the product. Because information is added to this file whenever updates are made to the program product, the history file reflects all changes made to the currently installed version and release of the program product. (2) A file containing a log of system actions and operator responses. (3) A file that displays all versions of a structured file.

IAR. See *instruction address register.*

I/O. See *input/output.*

ID. Identification.

immediate data. Data appearing in an instruction itself (as opposed to the symbolic name of the byte of data). The data is immediately available from the instruction and therefore does not have to be read from memory.

index. (1) A table containing the key value and location of each record in an indexed file. (2) A computer storage position or register, whose contents identify a particular element in a set of elements.

informational message. A message providing information to the operator, that does not require a response.

initial program load (IPL). The process of loading the system programs and preparing the system to run jobs. See *initialize.*

initialize. To set counters, switches, addresses, or contents of storage to zero or other starting values at the beginning of, or at prescribed points in, the operation of a computer routine.

input-output channel controller (IOCC). A hardware component that supervises communication between the input/output bus and the processor.

input-output code number (IOCN). A value supplied by the virtual machine to a VRM component. This number uniquely identifies the code associated with a component and can be considered a module name.

input-output device number (IODN). A value assigned to a device driver by the virtual machine or to a virtual device by the Virtual Resource Manager. This number uniquely identifies the device regardless of whether it is real or virtual.

input/output subsystem. That part of the VRM comprised of processes and device

managers that provides the mechanisms for data transfer and I/O device management and control.

instruction address register (IAR). A system control register containing the address of the next instruction to be executed. The IAR (sometimes called a "program counter") can be accessed via a supervisor call in supervisor state, but cannot be directly addressed in problem state.

integer. A positive or negative whole number or zero.

interactive. Pertains to an activity involving requests and replies as, for example, between a system user and a program or between two programs.

interface (n). A shared boundary between two or more entities. An interface might be a hardware component to link two devices together or it might be a portion of storage or registers accessed by two or more computer programs.

interrupt. (1) To temporarily stop a process. (2) In data communications, to take an action at a receiving station that causes the sending station to end a transmission. (3) A signal sent by an I/O device to the processor when an error has occurred or when assistance is needed to complete I/O. An interrupt usually suspends execution of the currently executing program.

invoke. To start a command, procedure, or program.

IOCC. See *input-output channel controller.*

IOCN. See *input-output code number.*

IODN. See *input-output device number.*

IPL. See *initial program load.*

kernel. The memory-resident part of the AIX Operating System containing functions needed immediately and frequently. The kernel supervises the input and output, manages and controls the hardware, and schedules the user processes for execution.

key pad. A physical grouping of keys on a keyboard (for example, numeric key pad, and cursor key pad).

load. (1) To move data or programs into storage. (2) To place a diskette into a diskette drive, or a magazine into a diskette magazine drive. (3) To insert paper into a printer.

local. Pertaining to a device directly connected to your system without the use of a communications line. Contrast with *remote.*

log. To record; for example, to log all messages on the system printer. A list of this type is called a log, such as an error log.

logical link control. In RT PC, a protocol process that allows data transfer on a given physical link. A logical link control (LLC) may reside in the operating system or the VRM and is controlled by the block I/O device manager.

loop. A sequence of instructions performed repeatedly until an ending condition is reached.

low-order. Least significant; rightmost. For example, in a 32 bit register (0-31), bit 31 is the low-order bit.

main storage. The part of the processing unit where programs are run.

mask. A pattern of characters that controls the keeping, deleting, or testing of portions of another pattern of characters.

menu. A displayed list of items from which an operator can make a selection.

minidisk. A logical division of a fixed disk that may be further subdivided into one or more partitions. See *partition.*

mm. Millimeter.

modem. See *modulator-demodulator.*

modulator-demodulator (modem). A device that converts data from the computer to a signal that can be transmitted on a communications line, and converts the signal received to data for the computer.

module. A discrete programming unit that usually performs a specific task or set of tasks. Modules are subroutines and calling programs that are assembled separately, then linked to make a complete program. See *object module.*

multiprogramming. The processing of two or more programs at the same time.

nonswitched line. A connection between computers or devices that does not have to be established by dialing. Contrast with *switched line.*

nonvolatile random access memory (NVRAM). A portion of random access storage that retains its contents after electrical power to the machine is shut off.

null. Having no value, containing nothing.

null character (NUL). The character hex 00, used to represent the absence of a printed or displayed character.

NVRAM. See *nonvolatile random access memory.*

object module. A set of instructions in machine language. The object module is produced by a compiler or assembler from a subroutine or source module and can be input to the linker. The object module consists of object code. See *module.*

octal. A base eight numbering system.

op code. See *operation code.*

operand. An instruction field that represents data (or the location of data) to be manipulated or operated upon. Not all instructions require an operand field.

operating system. Software that controls the running of programs; in addition, an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

operating system state. One of two virtual machine protection states that run in the processor's unprivileged state. The kernel is the only entity that runs in the operating system state. See *problem state.*

operation code. A numeric code indicating to the processor which operation should be performed.

overflow condition. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

pad. To fill unused positions in a field with dummy data, usually zeros or blanks.

page. (1) A block of instructions, data, or both.

page fault. A program interruption that occurs when a page that is not in memory is referred to by an active page.

page space minidisk. The area on a fixed disk that temporarily stores instructions or data currently being run. See also *minidisk.*

paging. The action of transferring instructions, data, or both between real storage and external page storage.

parameter. Information that the user supplies to a panel, command, or function.

partition. A logical division of a minidisk.

path name. See *full path name* and *relative path name.*

PEL. See *picture element*

picture element (pel). In computer graphics, the smallest element of a display space that can be independently assigned color and intensity.

PID. See *process ID.*

POST. See *power-on self test.*

power-on self test (POST). An internal diagnostic program activated each time the system is turned on.

priority. The relative ranking of items. For example, a job with high priority in the job queue will be run before one with medium or low priority.

privileged instructions. System control instructions that can only run in the processor's privileged state. Privileged instructions generally manipulate virtual machines or the memory manager; they typically are not used by application programmers. See *privileged state.*

privileged state. A hardware protection state in which the processor can run privileged instructions.

problem state. One of two virtual machine protection states that run in the processor's unprivileged state. User-written application programs typically run in the problem state. See *operating system state.*

process. (1) A sequence of actions required to produce a desired result. (2) An entity receiving a portion of the processor's time for executing a program. (3) An activity within the system begun by entering a command, running a shell program, or being started by another process.

process ID (PID). A unique number assigned to a process that is running.

program status block (PSB). A control block that describes a virtual interrupt condition.

protection state. An arrangement for restricting access to or use of all or part of the hardware instruction set. Hardware protection states are privileged state and unprivileged state. Virtual machine protection states are operating system state and problem state.

protocol. In data communications, the rules for transferring data.

protocol procedure. A procedure that implements a function for a device manager. For example, a virtual terminal manager may use a protocol procedure to interpret the meaning of keystrokes.

PSB. See *program status block.*

queue. A line or list formed by items waiting to be processed.

raster array. In computer graphics, a predetermined arrangement of lines that provide uniform coverage of a display space.

reentrant module. A module that allows the same copy of itself to be used concurrently by two or more tasks. Contrast with *serially reusable.*

register. (1) A storage area, in a computer, capable of storing a specified amount of data such as a bit or an address. Each register is 32 bits long. (2) See *general purpose register.*

relative address. An address specified relative to the address of a symbol. When a program is relocated, the addresses themselves will change, but the specification of relative addresses remains the same.

relocatable. A value, expression, or address is relocatable if it does not have to be changed when the program is relocated.

remote. Pertaining to a system or device that is connected to your system through a communications line. Contrast with *local.*

retry. To try the operation again that caused the device error message.

return code. A value that is returned to a subroutine or function to indicate the results of an operation issued by that program.

routine. A set of statements in a program causing the system to perform an operation or a series of related operations.

scatter. For input/output operations, reading data from a device and locating it in noncontiguous memory addresses. See *gather* (antonym).

SDT. See *static debugger trap.*

second level interrupt handler (SLIH). A routine that handles the processing of an interrupt from a specific adapter. An SLIH is called by the first level interrupt handler associated with that interrupt level.

sector. (1) An area on a disk track or a diskette track reserved to record information. (2) The smallest amount of information that can be written to or read from a disk or diskette during a single read or write operation.

segment. A contiguous area of virtual storage allocated to a job or system task. A program segment can be run by itself, even if the whole program is not in main storage.

semaphore. Entity used to control access to system resources. Processes can be locked to a resource with semaphores if the processes follow certain programming conventions.

serially reusable module. A module that can be accessed by only one task at a time. Contrast with *reentrant.*

session. (1) The period of time during which programs or devices can communicate with each other. (2) A name for a type of resource that controls local LU's, remote LU's, modes, and attachments.

single-step instruction execution. A method of operating a computer in which each instruction is performed in response to a single manual operation. No sequential execution of instructions is allowed.

SLIH. See *second level interrupt handler.*

special character. A character other than a letter or number. For example; *, +, and % are special characters.

stack. An area in storage that stores temporary register information and returns addresses of subroutines.

stack parameter. A parameter to a VRM subroutine that must be passed on the stack. Usually, VRM subroutines pass parameters in general purpose registers.

stack pointer. A register providing the current location of the stack.

static debugger trap (SDT). A trap instruction placed in a pre-defined point in code invokes the debugger at that point. The trap instruction causes a program check when executed; debugger is invoked as a result of the program check.

string. A linear sequence of entities such as characters or physical elements. Examples of strings are alphabetic string, binary element string, bit string, character string, search string, and symbol string.

subroutine. (1) A sequenced set of statements that may be used in one or more computer programs and at one or more points in a computer program. (2) A routine that can be part of another routine.

subsystem. A secondary or subordinate system, usually capable of operating independently of, or synchronously with, a controlling system.

supervisor call (SVC). An instruction that interrupts the program being executed and passes control to the supervisor so it can perform a specific service indicated by the instruction.

SVC. See *supervisor call.*

switched line. In data communications, a connection between computers or devices

established by dialing. Contrast with *nonswitched line.*

synchronous. (1) Pertaining to two or more processes that depend upon the occurrences of specific events such as common timing signals.

system customization. A process of specifying the devices, programs, and users for a particular data processing system.

system dump. A printout of storage from all active programs (and their associated data) whenever an error stops the system. Contrast with *task dump.*

thrashing. A condition in which the system is doing so much paging that little useful work can be done.

TLB. See *translation lookaside buffer.*

trace. To record data that provides a history of events occurring in the system.

track. A circular path on the surface of a fixed disk or diskette on which information is magnetically recorded and from which recorded information is read.

translation lookaside buffer (TLB). Hardware that contains the virtual-to-real address mapping.

trap. An unprogrammed, hardware-initiated jump to a specific address. Occurs as a result of an error or certain other conditions.

typematic key. A key that repeats its function multiple times when held down.

type declaration. The specification of the type and, optionally, the length of a variable or function in a specification statement.

typestyle. Characters of a given size, style and design.

unprivileged instruction. Ordinary instructions such as load store, add, and shift typically used by application programs.

unprivileged state. A hardware protection state in which the processor can only run unprivileged instructions. The processor's unprivileged state supports the virtual machine's operating system state and problem state.

update file. A file that adds or revises information in a program product already resident on the VRM minidisk. An update file documents the version, release, and level of updates to be installed and is required for program product update diskettes that use VRM Install update facilities. This file is not used for programs updated from the AIX Operating System.

variable. A name used to represent a data item with a value that can change while the program is running. Contrast with *constant.*

vector. An array of one dimension.

virtual device. A device that appears to the user as a separate entity but is actually a shared portion of a real device. For example, several virtual terminals may exist simultaneously, but only one is active at any given time.

virtual machine. A functional simulation of a computer and its related devices. A virtual machine usually includes an operating system and one or more virtual devices.

virtual machine interface (VMI). A software interface between work stations and the operating system The VMI shields operating system software from hardware changes and low-level interfaces and provides for concurrent execution of multiple virtual machines.

virtual memory manager. Hardware that manages virtual memory by providing translation from a virtual address to a real address.

virtual resource manager (VRM). A set of programs that manage the hardware resources

(main storage, disk storage, display stations, and printers) of the system so that these resources can be used independently of each other.

virtual storage. Addressable space that appears to be real storage. From virtual storage, instructions and data are mapped into real storage locations.

VMI. See *virtual machine interface.*

VRM. See *virtual resource manager.*

word. A contiguous series of 32 bits (four bytes) in storage, addressable as a unit. The address of the first byte of a word is evenly divisible by four.

Index

Special Characters

A

B

E

F

S

T

U

V

W

Numerics

IBM RT PC Programming Family

Reader's Comment Form

IBM RT PC Virtual Resource Manager Device Support

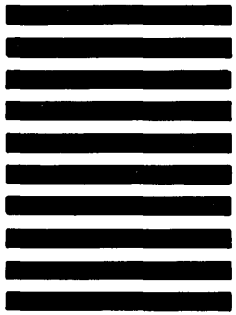
SC23-0817

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact the authorized IBM RT PC dealer in your area.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Fold and tape

Fold and tape

Cut or Fold Along Line

Tape

Please Do Not Staple

Tape

IBM RT PC VRM Device Support — **SC23-0817**

Book Title — **Order No.**

Book Evaluation Form

Your comments can help us produce better books. You may use this form to communicate your comments about this book, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Please take a few minutes to evaluate this book as soon as you become familiar with it. Circle Y (Yes) or N (No) for each question that applies and give us any information that may improve this book.

Y N Is the purpose of this book clear?

Y N Is the table of contents helpful?

Y N Is the index complete?

Y N Are the chapter titles and other headings meaningful?

Y N Is the information organized appropriately?

Y N Is the information accurate?

Y N Is the information complete?

Y N Is only necessary information included?

Y N Does the book refer you to the appropriate places for more information?

Y N Are terms defined clearly?

Y N Are terms used consistently?

Y N Are the abbreviations and acronyms understandable?

Y N Are the examples clear?

Y N Are examples provided where they are needed?

Y N Are the illustrations clear?

Y N Is the format of the book (shape, size, color) effective?

Other Comments

What could we do to make this book or the entire set of books for this system easier to use?

Optional Information

Your name _____

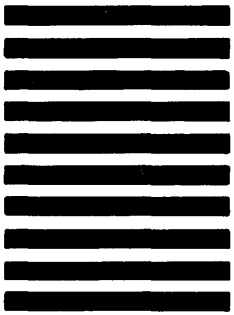
Company name _____

Street address _____

City, State, ZIP _____

No postage necessary if mailed in the U.S.A.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Fold and tape

Fold and tape

Cut or Fold Along Line

Tape

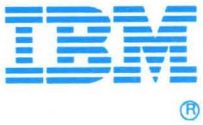
Please Do Not Staple

Tape

International Business
Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Printed in the
United States of America

SC23-0817-0



SC23-0817-00

92X1297