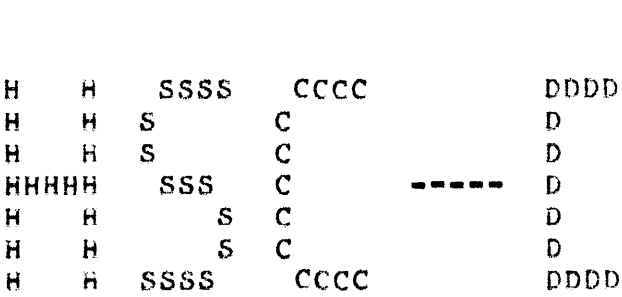
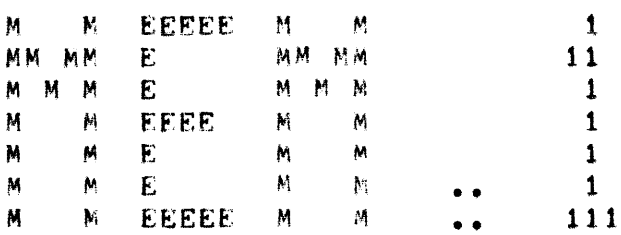



*START* Job HSC-DC Req #101 for DEUFEL.TL Date 9-Apr-82 10:16:56 Monitor: KL210
File PS:<DOC-SPECS>HSC-DCP.MEM.1, created: 11-Sep-79 11:26:39, printed: 9-Apr-82 1
Job parameters: Request created: 9-Apr-82 10:09:23 Page limit:246 Forms:NORMAL
File parameters: Copy: 1 of 1 Spacing:SINGLE File format:ASCII Print mode:ASC

This version of Chap 4 results from:

1. A new packet format from the Port and Protocol Task Force.

2. Host port id is now 32 bits.

3. Device class for the controller is DV.MSC for mass storage controller.

4. Minor revisions to the characteristics and status block formats.

5. Reserved fields must be zero.

An outstanding issue is the method to be used by a host to determine actual unit numbers since the controller characteristics includes only the number of units of each class.

The maintenance of this document will now be handled by Bill Grace.

Your comments will be appreciated by Bill.

Bob Mitchell

# CHAPTER 4

# PRELIMINARY HSC50 HOST INTERFACE SPECIFICATION

## Version 0.9

# 4.1 INTRODUCTION.

## 4.1.1 Scope.

This chapter defines the interface between the host computer and the HSC50 Controller.

Section 4.2 defines the general functional responsibilities which the Host and the Controller respectively have.

Section 4.3 specifies the constraints on the physical I/O interface by which the Host connects to the Controller.

Section 4.4 specifies the overall geometry of the information on the disk and the performance characteristics of the Controller as seen by the host.

Section 4.5 defines the mechanisms (protocol) by which control information is passed between the Host and Controller.

NOTE:

This specification is predicated and built upon four related documents. An understanding of the contents of these documents is recommended to the appreciation of this document.

DCP (DEVICE CONTROL PACKETS): A PROPOSED STANDARD FOR MASS STORAGE DEVICES, R. Supnik, Revision 2, 1-Oct-78

VAX-11 PORT ARCHITECTURE, W. Strecker, Revision 2, 22-Sep-78

DEC STD xxx, STANDARD FOR INTELLIGENT DISK BAD BLOCK REPLACEMENT, B. Rubinson, (in preparation, herein referred to as DEC Std 166)

ICCS BUS PROTOCOL, PRELIMINARY SPECIFICATION, W. Strecher, 26-Dec-78

## 4.1.2 Terminology.

4.1.2.1 Bus and Architecture Terminology -

1. "Subsystem" -- the subsystem is composed of the controller and all peripheral devices attached to and accessed through that controller.

2. "Controller" -- the controller is the hardware and software which communicates with the host via DCP and causes the commanded activity to occur on a unit.

3. "Device Classes" -- devices in the same device class share common functional characteristics. They transfer data in the same manner, are accessed similarly, and differ essentially only in their access times, transfer rates, geometries, and error characteristics. There are two classes of devices that are of concern to this chapter:

    1. "Disk Class Devices" -- any random access, block replaceable mass storage device containing data of fixed length blocks. This class includes rotating disks, Dectapes, TU58s, bubbles, CCDs, etc.

    2. "Tape Class Devices" -- any sequential access, variable length record mass storage device. The data on these devices typically cannot be selectively overwritten. Devices in this class include magtapes and TU60 cassettes.

4. "Class Driver" -- the code in the host operating system that controls the operation of a peripheral or set of peripherals of the same class. It typically serves to interface the device-independent logical I/O system to the peculiarities of a particular device or class of devices. It is at the device or class driver level that the interface described in this chapter will be implemented in most host operating systems.

5. "Port Driver" -- the code in the host operating system that manages the operation of an I/O bus port. It typically serves to interface the totally command-delivery-mechanism independent device or class driver to the peculiarities of a particular bus or device command delivery mechanism.

6. "Unit" -- as used in this document, a unit is any distinct, addressable component in the subsystem that serves a logically distinct function from other components in the subsystem. Typically they represent individual disk drives, tape drives, or individual disks in multi-unit drives, but the controller itself is a logical unit as well. It is to logical units that individual commands are directed.

4.1.2.2 Disk Format Terminology. -

1. "Block"-- On disk class devices, a block corresponds to a single, addressable data region of 512 or 576 bytes in length (depending on format). For this class of devices, all blocks are of the same length.

   On tape class devices, a block corresponds to a single data record on the tape. For this class of devices, a block may be of any length.

2. "Logical Block Number (LBN)"-- A number which identifies a particular data block's relative position in the total set of blocks made available to the host on a particular unit of a particular device. Each unit presents to the host a set of logically contigous blocks numbered from 0 to n-1, where n is the total number of blocks available on the unit. This number bears no necessary relationship to any physical placement of that block on the unit itself; LBN to physical address translation is a function of the controller alone. The number of blocks per cylinder is a characteristic of a unit.

3. "Replacement Block Number"-- A number which identifies the block on the disk for placing a block of data when the original disk block location is bad. The block is still accessed by referencing the LBN. The replacement block number is assigned by the HSC50 when a write with replacement command is given. The host must specify the RBN for the UDA50.

4. "DEC STD 166"-- A proposed replacement for DEC STD 144. It provides substantially more information than the current standard, and is necessary to accomodate substantial changes in disk format induced by new technologies. Specifically, it includes information describing the parameters of disk format, a bad block map, and a replacement block map that is considerably larger than existing versions.

4.1.2.3 Tape Format Terminology -

\To be specified\

4.1.2.4 Control Protocol Terminology. -

1. "Packet"-- A packet is a single, discrete, self-contained control message transmitted between the host and controller via an undefined message delivery mechanism. Packets transmitted from the host to the controller convey command information to the controller. Packets transmitted between the controller and

host convey status and operation result information to the host.

2. "Hard Initialization"-- The restarting of the controller as if the controller is being powered up for the first time. All context from previous operation is lost. Hard initialization results from physical conditions in the controller (power up, component failure) or from a bus-level action by a host.

3. "Soft Initialization"-- The resetting and resynchronization of the controller relative to a particular host. The controller continues running and context not involved with the initializing host is retained, but operations relative to the initializing host itself are aborted and communication with the host is resynchronized. Soft initialization results from a command by a host. Soft initialization is required because hard initialization cannot be used to resynchronize in a multi-host environment.

4. "Characteristics"-- Characteristics are those controller or unit parameters which govern operation of the controller or unit, affect the interpretation of and action on commands, and change relatively infrequently. Examples of characteristics are shadowing relationships, controller error thresholds, and tape unit density and speed settings.

5. "Status"-- Status is the set of variable physical conditions describing the current physical and logical state of the controller or unit. Status can change relatively frequently. Examples of status are the online/offline logical state of a unit, error flags, and the physical state of the spindle and panel switches.

6. "Host Timer" and "Command Timer"-- Logical timers used to detect possible failure of the host or controller. The "host timer" is a timer set by the controller to monitor activity on the part of the host and detect failure to utilize the logical connection to the controller over an extended period of time. The "command timer" is a timer maintained by the host to monitor controller activity and detect controller failure to complete a command in an acceptable time frame.

7. "Controller Offline" and "Controller Online" -- Logical states of the subsystem relative to the host. The subsystem may be in either of these two states relative to each host to which it is physically connected, and may be in different states to different hosts. The logical state that a controller is in governs the set of commands it will process and actions it will take on behalf of the host.

8. "Unit Offline", "Unit Available", and "Unit Online" -- Logical states of a particular subsystem unit relative to the host. Depending on the logical state of the subsystem relative to the

host, individual units attached to that subsystem may also be in one of three logical states relative to a host. The logical state of a unit governs whether or not it is available to process requests on behalf of the host.

9. "Shadowing"-- The duplication of a mass storage transaction on two units of the same device class. A shadowed unit is typically an image of another unit; it contains no history describing the steps by which the image was arrived at.

## 4.2 HOST & CONTROLLER FUNCTIONAL RESPONSIBILITIES

### 4.2.1 Responsibilities of Controller

#### 4.2.1.1 Minimum Integrity Diagnosis -

The controller is responsible for executing sufficient diagnostics locally to establish minimum operational and communication integrity each time the controller is powered up or hard initialized. Among the elements which must be tested are the controller processor, internal busses, internal memory, load device (TU58), and communication port to the outside world.

It is also the controller's responsibility to avoid signalling to the host that it is available to come online unless these diagnostics have been successfully executed.

#### 4.2.1.2 Notification of Controller Status Changes -

The controller is responsible for making available to a connected host notification of all spontaneous changes in controller status other than those resulting in controller failure. Status changes include changes in the availability or physical state of any of the units attached to the controller, non-fatal changes in the configuration or state of the controller itself, or any spontaneous initializations that occur in controller or drives.

The set of possible status changes that can take place in the subsystem will be divided into classes ranging from the most serious (controller spontaneous reinitialization) to the commonplace (drive changes such as write protect, pack unloaded, etc.). The controller is responsible for providing the host with a mechanism for directing which classes of status changes are to be reported and which are not. In addition to the class of error setting for each host ,the controller will provide for each host to specify receipt of either error log information which is controller spontaneous, or controller spontaneous and those related to that host's commands, or controller spontaneous and those related to all hosts.

The controller is responsible for notifying all hosts described in its internal configuration data base of its hard reinitialization.

4.2.1.3 Notification of Errors -

The controller is responsible for notifying the host when a command cannot be successfully completed, and for supplying information regarding the exact nature of the error as part of that notification.

The controller is responsible for making available to the host appropriate information for the host error log whenever a command cannot be successfully completed or whenever a command is successfully completed, but the successful completion of the command required the use of an error recovery procedure in the subsystem.

The set of possible error log situations will be divided into classes ranging from the serious (drive mis-seek, ECC error involving large number of symbols) to the commonplace (single-symbol ECC error, physical bad block). The controller is responsible for providing the host with a mechanism for directing which classes of error log information are to be reported and which are not.

4.2.1.4 Command Integrity Verification -

The controller is responsible for verifying the integrity (opcode/parameter validity, etc.) of each request that it receives from the host, and for rejecting those commands that are not valid in the current subsystem context (as determined by the controller parameter base and state, i.e. current CHARACTERISTICS and STATUS).

The controller will execute each valid command, and has NO responsibility relative to the validity of a series of commands.

4.2.1.5 Verification of Host Connection Before Relinquishing Drive -

If a dual-ported drive 'online' to a controller requests verification of the drive-to-controller connection, the subsystem is responsible for verifying that it is still 'controller online' to at least one host before retaining its connection to the drive. A controller must not keep a dual-ported drive online if all hosts to which it can be logically connected are 'controller offline'.

4.2.1.6 Shadowing Responsibilities -

The controller is responsible for duplicating all write operations to a shadowed unit on the appropriate shadow unit, and for avoiding notification of successful completion of the initial operation unless and until both the initial operation and the duplicate are successfully

completed.

If a physical status change occurs for reasons other than host command, such as FE initiated pack spin down, the host will be notified on the next host reference to either unit or by an ATTENTION message, depending on the situation.

#### 4.2.1.7 Guarantee of FIFO Data Integrity on Disk Class Devices -

The controller is responsible for satisfying all WRITE operations of the same priority class to a particular LBN on a particular disk-class unit such that all operations of the same priority class on that LBN that preceeded it are completed before the WRITE is processed, and any operations of the same priority class to the same LBN that follow it are postponed until the WRITE is completed.

Note that this responsibilitility is within request priority classes only; the controller has no responsibility to satisfy requests of differing priority to the same LBN in any particular order, regardless of the fact that they may include conflicting reads and writes.

The controller is NOT responsible for satisfying requests to different LBNs on the same disk-class unit in any particular order, or for satisfying READ operations in any particular order.

#### 4.2.1.8 Guarantee of Serial Command Execution on Tape Class Devices -

The controller is responsible for executing all operations on tape class units serially in the order in which they are received.

#### 4.2.1.9 Timeout of Requests for Host Data -

The controller is responsible for timeout of each request for host data (for WRITE or COMPARE) sent to the host, and for suitable error recovery (to be defined) should that timeout actually occur.

#### 4.2.1.10 Change to WRITE-PROTECT Status -

If the controller can determine that a host is not operating correctly, the controller is responsible for changing its status relative to that host to WRITE-PROTECT. Normal operation can be reestablished by the host resynchronizing with the controller.

The criteria that the controller will use to establish that a host is disabled are enumerated in Section 4.5.2.1.2.1

4.2.1.11 Formatting and Reformatting -

The formatting and reformatting of units is the responsibility of the controller via software executing in the controller itself. Formatting and reformatting operations are initiated in the controller via the appropriate host or FE panel command.

4.2.1.12 Automatic Revectoring of Bad Blocks -

The controller is responsible for automatically revectoring bad blocks when formatting a volume and when copying one volume to another. An unrevectored bad block on the source unit will be copied and the data marked invalid on the copy.

4.2.1.13 Maintenance of DEC STD 166 Information -

The controller is responsibile for maintenance of and utilization of the information in the 166 area for the purpose of revectoring bad blocks.

The controller is also responsible for denying the host write access to the 166 area.

The host can utilize the information in the 166 area as appropriate to its particular needs. For example, the host can use the bad block and replacement block information in the 166 area as part of its file allocation policy if that policy requires such information.

The host accesses the 166 area directly via that portion of the LBN space assigned to that purpose on each unit.

4.2.2 Responsibilities of Host

#### 4.2.2.1 Minimum Interface Integrity Diagnosis -

The host is responsible for diagnosing the physical interface between itself and the controller. The host is responsible for executing this diagnostic before attempting to use the logical connection with the controller.

#### 4.2.2.2 Resynchronizing with the Controller -

The host is responsible for resynchronizing with the controller under the following circumstances:

1. The host itself is powered up or reinitialized.
2. The host is recovering from possible controller failure.

#### 4.2.2.3 Integrity of Commands -

The host is responsible for sending only syntactically and semantically valid commands to the controller. Furthermore, the host is responsible for the relationships BETWEEN commands; the host must not issue commands that conflict with each other or with commands from another host connected to the controller.

#### 4.2.2.4 Managing Controller Operation -

As part of managing the controller operation, the host is responsible for determining the current characteristics of the subsystem, and for changing the controller characteristics and thresholds to those necessary for proper operation with the connecting host.

The subsystem will operate under a set of default characteristics and thresholds should the host choose not to alter them, but in this case the host is responsible for operating meaningfully with the default subsystem characteristics.

#### 4.2.2.5 Unique Reference Numbers -

The host is responsible for supplying a UNIQUE reference number in every command that is outstanding to the controller at any given time.

4.2.2.6 Timeout of Commands -

The host is responsible for ensuring that each command that is transmitted to the controller is covered by a timer. If more than one command is simultaneously outstanding to the controller, then the host is responsible for covering a series of commands with a gross timer or for managing multiple timers. If any command timer expires, the host is responsible for appropriate error recovery action (defined in Section 4.5.2.4) The host should NEVER ask the controller to perform an untimed operation.

4.2.2.7 Reissue of Outstanding Commands -

Under certain error conditions (command timeout, controller reinitialization due to non-critical component failure, etc.) the host is responsible for resynchronizing with the controller and reissuing all commands that were outstanding at the time the error was detected. In this instance, the host is also responsible for insuring that the commands are not reissued until the prescribed resynchronization has been successfully accomplished. Note that across resynchronizations and reinitializations, the controller's set of outstanding commands are "erased", so the same reference number can be reused.

4.2.2.8 Error Recovery with a Failing Controller -

If a host detects that a controller is failing, it is the host's responsibility to cease issuing further commands to the controller and to execute the error recovery sequence prescribed in Section 4.5.2.4.

The criteria that the host is to use to determine that a controller is failing are enumerated in Section 4.5.2.4.

4.2.2.9 Revectoring of Wear-caused Bad Blocks -

When a bad block is reported to the host, the host is responsible for directing the controller to replace that bad block with a replacement block when and if consistent with host policy.

The controller will not replace bad blocks (unless formatting or copying a volume) unless directed to do so by the host.

4.2.2.10 Coordination with Other Host(s) -

In an environment where multiple hosts are connected to the same controller, the hosts are responsible for coordinating their access to the subsystem so that there is no conflict for the same units or data. The controller will satisfy any request for any unit from any host; it will NOT protect multiple hosts from each other.

Furthermore, the hosts must coordinate their subsystem characteristics changes such that conflicting characteristics commands are not given to the controller. The latest set of characteristics and parameters in force in the controller will be the latest set received from any host; the controller will not detect conflicts between multiple hosts requesting different characteristics or status parameters.

4.2.2.11 Shadowing -

If units are to be shadowed, the host is responsible for directing the controller which units are to be shadowed by which units as part of the controller characteristics establishment.

The host is responsible for any shadowing at other than the unit level by issuing the appropriate combination of individual commands to achieve the desired shadow functionality.

4.2.2.12 Error Logging -

It is the host's responsibility to process error log information passed to it from the controller in a manner consistent with host policy.

If error log information is to be reported to the user, it is the host's responsibility to do so.

The host is also responsible for determining and setting which classes of errors are to be reported by the controller.

## 4.3 HOST PHYSICAL INTERCONNECT

This specification discusses the functions that the device driver performs to control the operation of the controller. It is documented independent of whatever physical delivery mechanism is used to move command packets and data between the host and controller. The protocol described in this chapter assumes certain characteristics that the delivery mechanism must have, however, and these assumptions are documented in this section. Theoretically, any delivery mechanism which provides the necessary guarantees and functional interfaces should be usable with any disk or tape class driver that conforms to this specification. A new port driver for the new delivery mechanism should be all that is required.

### 4.3.1 Guarantees Required of Control Packet and Data Transport Mechanism

The delivery mechanism must be capable of guaranteeing the following to the device class driver:

1. That message packets queued to the port driver on a given priority level will be sent and received in the order in which they are queued.

2. That transmission errors will be detected and the appropriate low level error recovery operations (resynch, retry, etc.) will automatically occur.

3. That transmission errors reported by the port are not recoverable.

4. That message transmission will stop when an unrecoverable error is detected; that is, those messages queued to the port driver behind the erroneous message will not be transmitted until the class driver has intervened. This is required in order for the controller to be able to fulfill its LBN FIFO guarantee.

5. That a bus-level hard initialize function exists, and when invoked causes an unconditional reaction on the initialized node that forces the initialized node to a known state and does not depend on functioning software in the initialized node to operate.

6. That a bus-level echo function exists that completes a circuit through the bus and receiving node as well as the sender.

7. That non-acceptance of a message at a receiving port be recognizable at the sending port.

### 4.3.2 Functional Interface to Control Packet Mechanism

The interface between the port driver and the class driver is assumed to have the following functional characteristics:

1. A message is sent to the controller by passing to the port driver the address of a buffer containing the message and an indication that it is to be sent to a particular node on the bus (the controller).

2. The class driver is notified that an attempt to send a message was unsuccessful when the port driver passes back to the class driver the address of the buffer containing the original message and an indication that a transmission error has occurred. In this instance, further transmission will be inhibited until the class driver indicates directly that transmission is to resume.

3. The class driver is notified that a message has been received when the port driver passes to the class driver the address of a buffer containing the message and information defining the sender.

4. The class driver has control over whether the port is actively transmitting, receiving, or both (relative to the calling class driver only) via appropriate indications to the port driver.

5. The class driver has control over reinitialization and resynchronization of the port (relative to the calling class driver only) via appropriate indications to the port driver. When told to reinitialize, the port driver discards any outstanding work relative to the calling class driver and resets to an idle, non-transmitting and non-receiving state for that class driver ready to begin the first operation indicated by that class driver.

6. The class driver has the capability of causing a bus-level hard initialize of the specified node via appropriate indications to the port driver.

7. The class driver has the ability to verify the integrity of its connection with a specified node by passing the address of a buffer to the port driver and indicating that it is to be echoed via the appropriate node. The port driver will return the address of another buffer containing the echoed data when the echo loop has been completed.

### 4.3.3 Functional Interface to Data Transfer Mechanism

\To Be Specified\

## 4.4 CONTROLLER PHYSICAL CHARACTERISTICS

### 4.4.1 Geometry of Mass Storage Information

See SDI document(Chapter 5 of HSC50 Specification)

### 4.4.2 DEC STD 166 Areas

See 166 document.

### 4.4.3 Controller Performance Characteristics

See Chapter 3 of HSC50 Specification

## 4.5 CONTROL PROTOCOL (HSC50 Implementation of DCP - DEVICE CONTROL PROTOCOL)

### 4.5.1 General Characteristics of Control Protocol

The HSC50 control protocol has the following general characteristics:

1. The total HSC50 control interface is divided into four logical levels. The lower levels (0 & 1) are documented in the port specification. The higher levels (2 & 3) are documented in the following sections. The levels are:

    1. Level 0 - the physical system used to transmit information along the bus connecting the host and controller.

    2. Level 1 - the port system by which messages and data are transmitted between the host and controller.

    3. Level 2 - the actual control information passed between the host and controller. This level contains the messages themselves.

    4. Level 3 - fixed sequences of level 2 commands and responses that are executed as a group to perform higher logical functions.

2. It is a master/slave protocol. The HSC50 is a middle element in the host-drive communication, and as such is the servant of the host. The responsibilities in the protocol are asymmetric, allowing the host to dominate operations such as communication resynchronization.

3. The lower levels of the protocol are symmetric. Although the responsibilities in the protocol are asymmetric, the mechanisms used to dispatch these responsibilities are identical in both directions. The same packet format is used in both directions (with different semantic interpretations), and symmetric delivery mechanisms are assumed.

4. The control protocol is asynchronous. The host may have multiple commands outstanding to the controller at any given time, and the communication medium may be full duplex. Furthermore, the controller may send an undetermined number of unsolicited messages at any time.

### 4.5.2 General Rules Governing Control Protocol Usage

#### 4.5.2.1 Subsystem Logical States -

##### 4.5.2.1.1 Subsystem Logical State Definitions -

The subsystem may be in either of two logical states relative to any host. Note that in a multi-host environment, the subsystem logical state relative to a particular host is independent of its state relative to any other host. These states are:

1. "CONTROLLER OFFLINE". The 'controller offline' state is just what it implies. A subsystem that is 'contoller offline' is not available to the host and cannot perform any operations on behalf of the host. Note that there are several reasons a subsystem may be 'controller offline' to a given host, only some of which are inoperable hardware. The subsystem may be 'controller offline' to the host because the host bus enable button on the front panel has disabled the host bus, or it may be 'controller offline' because the operator or FE has removed the controller from service (via its FE panel).

   A host can distinguish a subsystem that is in the 'controller offline' state because it will not respond correctly to any command.

   Each time the subsystem makes the transition from the 'controller offline' state to the 'controller online' state, it sends a REINITIALIZED attention message to all hosts that are described in the controller's internal configuration tables. It will send this message regardless of whether the reinitialization was spontaneous or forced. After the REINITIALIZED message the next command from the host must be an INIT to assure proper synchronization of the host and the controller.

   The PSUEDO TERMINAL IN command is the only command which is potentially recognizable by the controller in the offline state and then only for diagnostics.

2. "CONTROLLER ONLINE". A subsystem that is 'controller online' to a host is logically connected to that host and can execute its entire command repertoire on behalf of that host.

   A host can distinguish a subsystem that is in the 'controller online' state because it will execute all commands and the summary status in the END packet will indicate that the current subsystem state is 'controller online'.

Note that the logical states of the subsystem are distinct from but related to the logical states of any units connected to the controller. If a subsystem is 'controller offline' to a host, then all units connected to the controller are also 'unit offline' to the host. If a subsystem is 'controller online', however, then the individual units connected to that controller may themselves be 'unit online', 'unit available', or 'unit offline' to the host, independent of each other, depending on their states relative to the controller. All state changes in the subsystem (including unit state changes) are reported to the host if the subsystem is 'controller online' and such information is desired.

4.5.2.1.2 Subsystem State Transitions -

4.5.2.1.2.1 Controller Offline -

The subsystem enters the 'controller offline' state relative to a particular host whenever it ceases to function because of power failure, hardware failure, software failure, spontaneous hard reinitialization, or commanded hard reinitialization. In addition, the subsystem enters the 'controller offline' state relative to the host after sending a DISCONNECTING attention message because the host bus enable switch on the controller panel has been disabled or a command removing the subsystem from service was received at the FE panel. Basically, any condition which disables the controller or destroys its context takes it into the 'controller offline' state.

4.5.2.1.2.2 Controller Online -

The subsystem enters the 'controller online' state relative to a particular host when it successfully executes the diagnostics after a power-up, bus level initalization or being returned to service by FE panel command.

The subsystem leaves the 'controller online' state relative to a particular host whenever the following occurs:

1. It is forced into the 'controller offline' state by bus-level init command, internal failure, FE command removing the subsystem from service, power failure or a host bus switch change.

Note that if the controller is forced down by host bus switch change, FE command removing it from service, or an internal condition that does not

impact its integrity, it tries to leave the 'controller online' state in an orderly fashion. This orderly disconnect is merely a convenience; the protocol is designed to function properly if orderly disconnect fails. See Section 4.5.2.5.2.

There is a special situation when the controller will change its status to WRITE PROTECT relative to a host. This occurs when the controller is in the 'controller online' state and concludes that the host is potentially down. The controller will conclude that the host is potentially down (not operating correctly) if any of the following occurs:

1. Unrecoverable bus errors are repeatedly reported by the port.

2. A request for host data repeatedly times out.

3. The host does not issue a command within the host timeout period specified in the GET CHARACTERISTICS command.

4. The controller receives \ to be specified number\ consecutive erroneous commands from the host.

### 4.5.2.2 Unit Logical States Relative to Host(s) -

#### 4.5.2.2.1 Unit Logical State Definitions -

Each individual unit may be in one of three logical states relative to each host. The unit logical state is a function of the subsystem logical state relative to the host, and the unit's logical state relative to the controller itself. Units have logical states independent of each other. Furthermore each unit occupies the same logical state relative to each host that has the same logical state relative to the subsystem. For example, units that are online to the controller are 'unit online' to all hosts to which the subsystem is 'controller online'.

The unit states relative to the host are:

1. "UNIT OFFLINE". A unit that is 'unit offline' is one that is unavailable to satisfy any host requests. Units may be 'unit offline' due to hardware failure, an FE command removing the unit from service, or an inability to come online to the controller (e.g., it may already be online to another subsystem and is hence offline to this one).

   The subsystem will reject requests for a unit that is 'unit offline' to the host. The host can determine that a unit is 'unit offline' by interrogating its status.

2. "UNIT AVAILABLE". A unit that is 'unit available' is one that is available to come online to the host, but has not yet done so because no commands have yet been received for that unit. 'Unit available' is a temporary state that is significant to the host only because notification of transition to this state often serves as a key that operations to the unit can begin.

   The subsystem will attempt to satisfy requests to a unit that is 'unit available' by attempting to bring the unit online to the controller. The host can determine that a unit is 'unit available' by interrogating its status.

3. "UNIT ONLINE". A unit that is 'unit online' to the host is one that is online to the controller and can execute functions on behalf of the host.

   The subsystem will accept and execute commands for a unit that is 'unit online'. The host can determine that a unit is 'unit online' by interrogating its status.

#### 4.5.2.2.2 Unit Logical State Transitions -

##### 4.5.2.2.2.1 Unit Offline -

An individual unit enters the 'unit offline' state relative to the host whenever any of the following occurs:

1. The subsystem leaves the 'controller online' state.

2. The unit becomes offline to the controller; that is, the unit is not available for controller access. See the next chapter for drive states relative to the controller.

3. The unit is removed from service (taken 'unit offline' from the host) by a command at the FE panel or by a diagnostic internal to the subsystem.

A unit leaves the 'unit offline' state whenever the subsystem is in the 'controller online' state and any of the following occurs:

1. The unit reinitializes and becomes available to the controller. See the next chapter for drive states relative to the controller.

2. The unit is returned to service (made 'unit available' or 'unit online' to the host) by a command at the FE panel or by a diagnostic internal to the subsystem.

3. The unit's status is successfully changed to 'unit available' or 'unit online' by a host SET STATUS command.

A unit also leaves the 'unit offline' state whenever the subsystem enters the 'controller online' state and:

1. the unit is available to the controller, in which case it enters the 'unit available' state, or

2. the unit is online to the controller, in which case it enters the 'unit online' state.

4.5.2.2.2.2 Unit Available -

A unit enters the 'unit available' status relative to the host whenever the subsystem is 'controller online' and one of the following occurs:

1. The drive signals its availability to the controller.

2. The drive is in the available state relative to the controller and is returned to service by FE or subsystem diagnostic command.

A unit also enters the 'unit available' state when the subsystem enters the 'controller online' state and the drive is in the available state relative to the controller.

A unit leaves the 'unit available' state whenever one of the following occurs:

1. The subsystem leaves the 'controller online' state.

2. The subsystem receives a command for the unit and brings the unit online to the controller.

3. The host changes the unit status to 'unit online' via a SET STATUS command.

4.5.2.2.2.3 Unit Online -

An individual unit becomes 'unit online' to the host whenever the subsystem is in the 'controller online' state and one of the following occurs:

1. The controller receives a valid command for a 'unit available'

unit and succeeds in bringing that unit online to the controller.

2. The unit is returned to service (placed 'unit online' to the host) by a command at the FE panel or by a diagnostic internal to the subsystem.

3. The unit's status is successfully changed to 'unit online' by a host SET STATUS command.

A unit also enters the 'unit online' state whenever the subsystem enters the 'controller online' state and the unit is online to the controller.

An individual unit leaves the 'unit online' state relative to the host whenever one of the following occurs:

1. The subsystem leaves the 'controller online' state.

2. The drive leaves the online state relative to the controller. See next chapter.

3. The unit is removed from service by an FE command or by a diagnostic internal to the subsystem.

4. The unit's state is changed to 'unit available' by a host SET STATUS command.

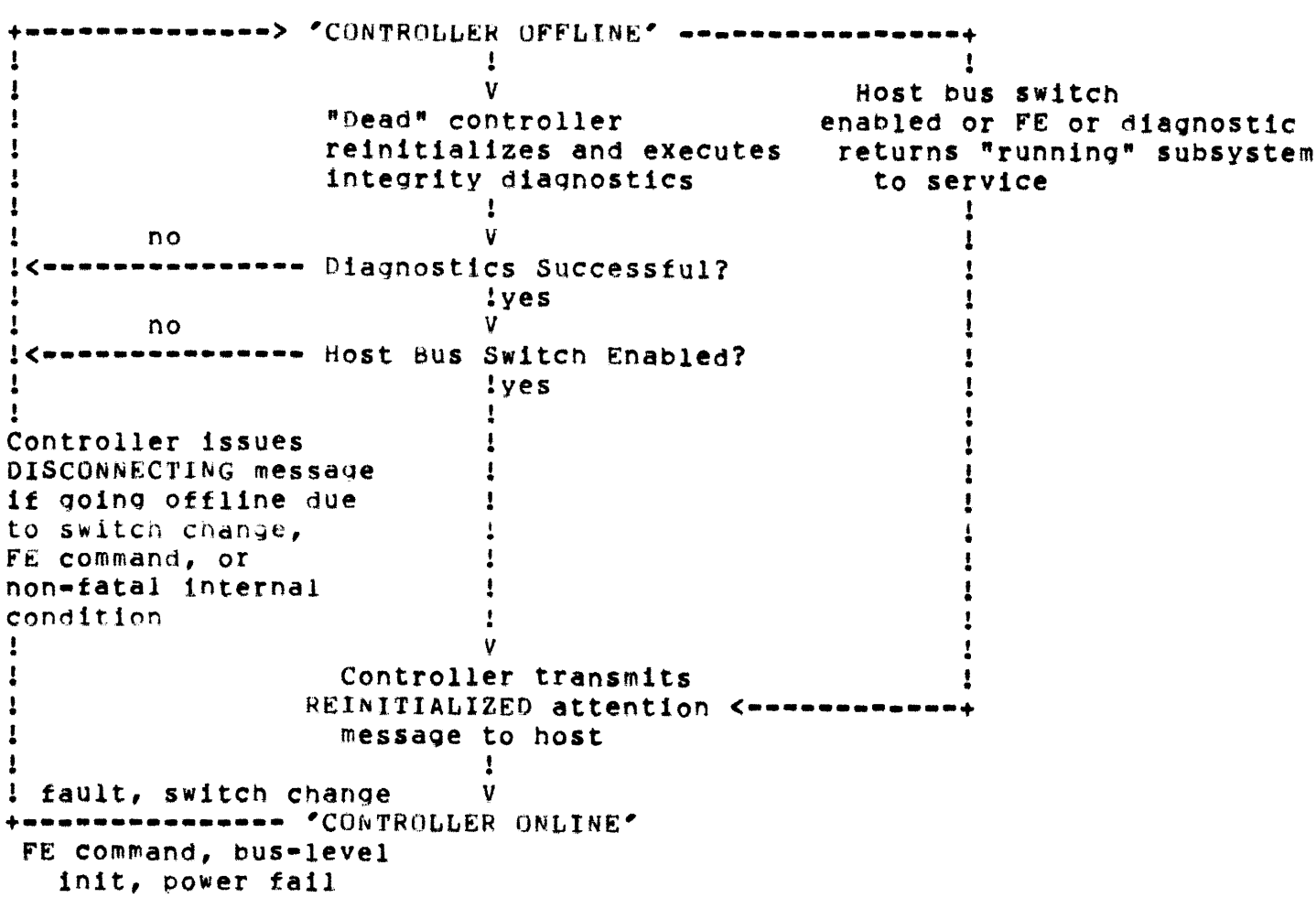
## RELATIONSHIP OF CONTROLLER AND UNIT STATES

The states in the boxes are the unit states relative to the host:

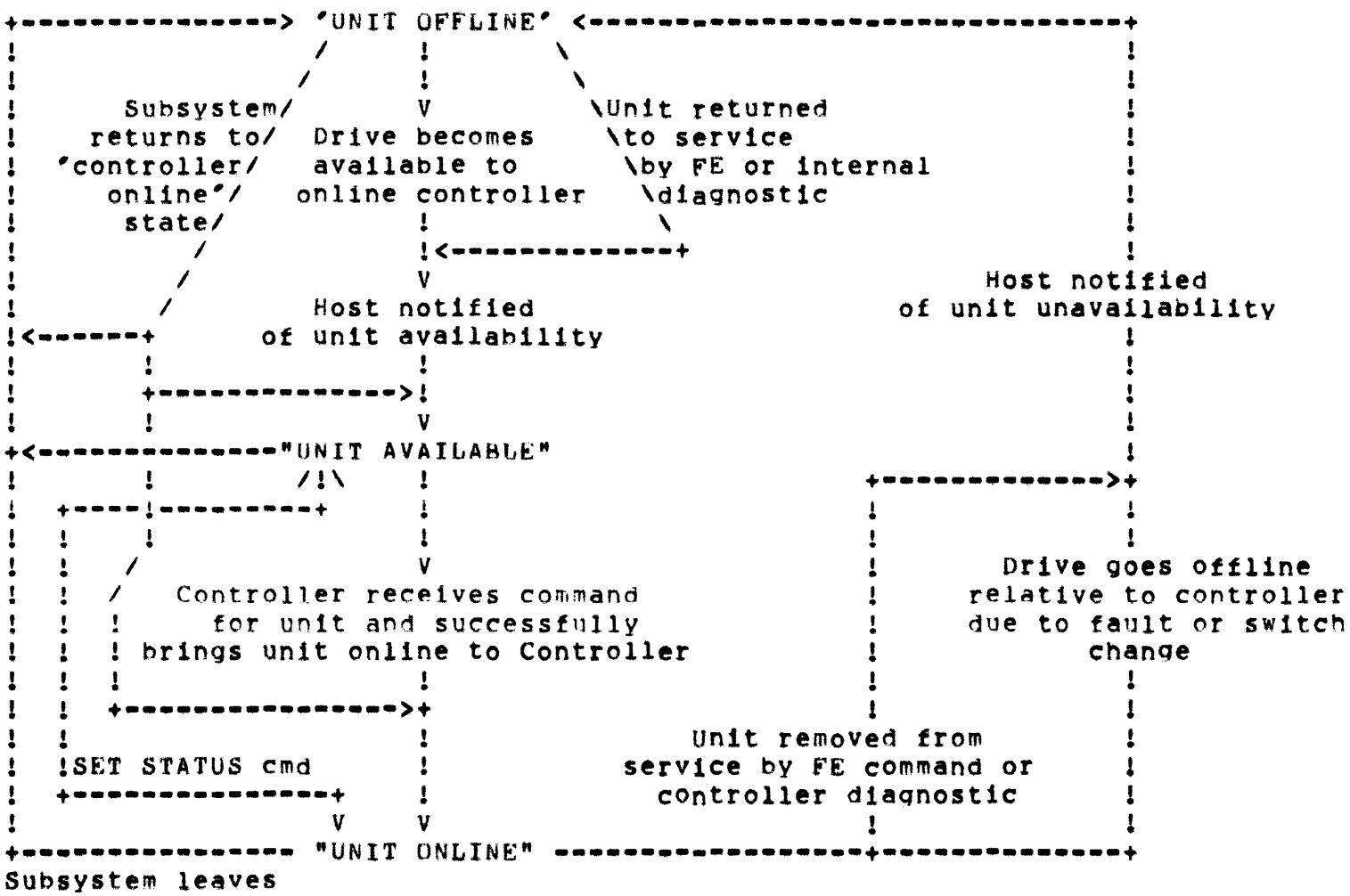
| Drive State Relative to Controller | Subsystem State Relative to Host: 'Controller Offline' | Subsystem State Relative to Host: 'Controller Online' |
|---|---|---|
| 'Offline' | 'Unit offline' | 'Unit offline' |
| 'Available' | 'Unit offline' | 'Unit available' |
| 'Online' | 'Unit offline' | 'Unit online' |

In general, whenever the subsystem enters the 'controller online' state, the unit enters the state dictated by its state relative to the controller. Also, a unit's logical state is only visible and variable when the controller is 'online'.

## SUBSYSTEM LOGICAL STATE TRANSITIONS RELATIVE TO A SINGLE HOST



UNIT LOGICAL STATE TRANSITIONS RELATIVE TO A SINGLE HOST



4.5.2.3 Command Flow -

The normal flow of each command issued by the host is diagrammed below. Basically, the command is issued and timing for that command is started. The controller will interpret the command, initiate any data transfer operations that are required to complete the command, and signal the eventual completion of the command (whether successful or not) with an END message. At any time a unit, including the controller, may send a LOG message; if the log packet is related to a particular command, it will contain the reference number of the command in question. If it is a general log message not related to any particular command, the reference number will be 0.

For transfers larger than one sector, the controller is likely to divide that transfer into smaller sub-transfers, known internally as "fragments". The data for each fragment of a WRITE transfer will be fetched from the host via a seperate request. The order in which fragments are processed will depend on conditions internal to the subsystem and frequently will not be in order of ascending LBN or buffer address.

Note that attention messages may also be sent at any time relating to any unit in the subsystem.

Note also that the steps in this flow may not necessarily be contiguous; if more than one command is outstanding, message and data traffic relating to any or all of these commands may be interleaved in a totally random fashion.

READ COMMAND FLOW

f = number of fragments necessary
to successfully complete transfer

| | |
|---|---|
| Step 1 | Host sends READ command packet requesting read of n blocks and starts timing command. |
| Step 2 | Controller sends data for a fragment to host memory via bus data transfer mechanism. |
| . | . |
| . | . |
| . | . |
| Step f+1 | Controller sends data for last oustanding fragment to host memory via bus data transfer mechanism. |
| Step f+2 | Controller sends END packet signalling completion of command and indicating status. Host stops timing command. |

## WRITE COMMAND FLOW

f = number of fragments necessary
to successfully complete transfer

| | |
|---|---|
| Step 1 | Host sends WRITE command packet requesting write of n blocks and starts timing command. |
| Step 2 | Controller requests host port to send data for a fragment from host memory via bus data transfer mechanism. |
| Step 3 | Host sends data for requested fragment via bus data transfer mechanism. |
| | . . . |
| Step 2f+1 | Controller requests host port to send data for last oustanding fragment from host memory via bus data transfer mechanism. |
| Step 2f+2 | Host sends data for last outstanding fragment via bus data transfer mechanism. |
| Step 2f+3 | Controller sends END packet signalling completion of command and indicating status. Host stops timing command. |

4.5.2.4 Error Recovery -

The host has only two basic error recovery procedures available to it for errors at Level 2 and Level 3. Level 0 and Level 1 errors are dealt with by the Transport Mechanism. If a command fails because of a physical error, the subsystem has already taken all corrective action possible, and further recovery attempts by the host are useless. If a command is rejected because of a logical inconsistency, the host corrects the problem and reissues the command. If a command fails for any other reason, the host must resynchronize with or reinitialize the controller, and reissue all outstanding commands at the time of the failure, including the failing command. Note that the details of the resynchronization and reinitialization procedures are described in the sections that follow.

Particulars of these error situations follow:

1. If the controller reports that it cannot execute a valid command for a particular reason, the host is to take error recovery action appropriate to the indicated reason. For example, if a valid command is rejected by the controller, the host should attempt to resynchronize with the controller. If the controller reports that a hard I/O error has occured on a request, the host is to assume that the controller has taken all corrective action possible, and is to report the error to the application as uncorrectable.

2. If the command timer for any particular command expires, the host is to issue a GET STATUS command as a means of verifying the bus and connection. If this GET STATUS command is not successfully sent by the port, the connection to the controller should be assumed broken. If the GET STATUS is successfully sent and either succeeds or times out, the host should set a longer INIT timer and issue an INIT command to the controller to resynchronize and flush outstanding commands. If the INIT succeeds, the host is to reissue all outstanding commands, including the one that timed out originally. If the INIT fails or times out, the host is to set a very long hard init timer and issue a bus-level init to the controller. If the hard init timer also times out before a REINITITIALIZED attention message is received from the controller, the host is to assume the controller is down and wait until the message is received (it always will be received eventually). If the hard init succeeds, the host is to reconnect to the controller and reissue the commands that are outstanding, including the original one that timed out.

   Note that the timers that are set for the INIT and bus-level init operations must be \to be specified\ and \to be specified\ multiples of the standard command timer respectively. This is necessary to avoid race conditions caused by multiple-hosts issuing hard inits at close intervals.

A diagram of the command timeout recovery sequence is at the end of this section.

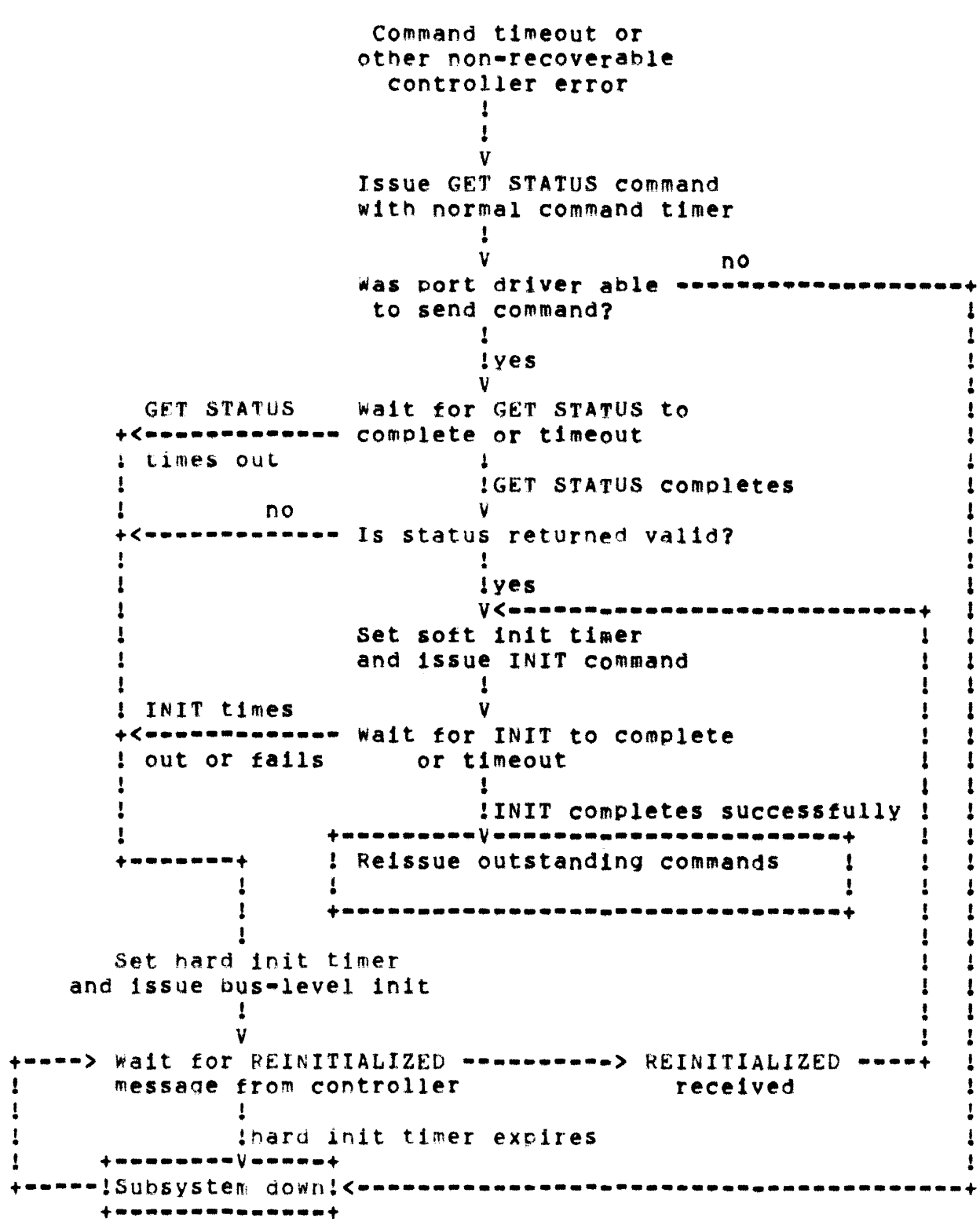
3. If the controller reports an error that could be a communications error (invalid command code, invalid args, etc.) and the command appears valid to the host, the host may treat the error as if the offending command timed out or may try retransmission, depending on the reliability of the transport mechanism.

4. If the host receives a communication from the controller that it cannot interpret (jibberish) or that is obviously out of context (e.g., an END for a command that cannot be outstanding), the host should treat the error as if a command timed out.

5. If the host is restarted, it must resynchronize with the controller.

6. If the host receives a REINITIALIZED attention message from the controller or a command is rejected because the subsystem has a WRITE-PROTECT status relative to the host, the host must resynchronize with the controller and reissue outstanding commands.

FAILED COMMAND RECOVERY SEQUENCE



4.5.2.4.1 Resynchronization -

The host is the master in the host-controller relationship, and as such it is responsible for synchronization with the controller. Whenever there is a question as to the state of either party across the interface, the host must resynchronize as described below. The controller's responsibility is limited to notifying the host of the need for resynchronization.

Note: the synchronization described here is for the highest levels of the protocol only. Any resynchronization at lower levels is independent of this level and is the responsibility of the port and port drivers. The levels described here assume an already synchronized transport mechanism.

Note that resynchronization will not always succeed in restoring a malfunctioning controller to a fully operational state. It will succeed much of the time, however, and it is recommended because it is much faster that hard reinitialization, and unlike hard reinitialization, it does not affect other hosts.

The steps involved in resynchronization are as follows:

1. The host ceases issuing further commands to the controller.

2. The host sets a soft init timer and issues a INIT command to the controller and waits for the END response. When the controller receives this command, it will cease processing on all outstanding host requests that it can effectively stop, and will discard those commands. Commands that are too far along to be stopped will be finished and will terminate with normal END messages.

3. The controller responds with an END message to the INIT command.

4. If any outstanding host requests remain not completed when the END for the INIT command is received, the host reissues these requests as appropriate.

4.5.2.4.2 Reinitialization -

As master, the host is also responsible for reinitializing the controller under certain circumstances.

Note that since reinitialization involves a cold bootstrap of the controller, ALL controller context relative to ALL hosts is lost during this process. Reinitialization is thus a last ditch action taken when any other host(s) are attached to the same controller. Since it is also

a time-consuming and wear-causing process, it must not be part of routine error recovery.

Reinitialization consists of forcing the controller to perform a cold restart, then resynchronizing with it once (and if) it comes back up.

The steps involved in reinitialization are as follows:

1. The host ceases issuing further commands to the controller.

2. The host sets a hard init timer and issues a bus-level init command to the controller. It then waits for a REINITIALIZED command from the controller. As soon as the bus-level init is recognized, all perceivable activity in the controller will stop relative to all hosts.

3. The controller will perform a cold bootstrap as if it was being powered up. When and if it passes its minimum integrity diagnostics, it will send a REINITIALIZATION message to all hosts in its configuration tables. The controller is then online to all hosts.

4. If host requests were outstanding when the reinitialization process was begun or the REINITIALIZED message is received, a host reissues these requests as appropriate.

#### 4.5.2.5 Connection Establishment and Termination -

The establishment of a logical connection between the host and the controller is automatic when the controller changes to the online state for all hosts in its configuration table. If the host is to use the subsystem in an optimal manner, the Configuration Sequence must be executed to determine the exact status of the subsystem and to set any characteristics and/or status to other than the default values. The conection exists, from the controllers viewpoint, until the controller odnegoes offline. This may be done by a host during SYSGEN or dynamically during each boot of the host.

##### 4.5.2.5.1 Configuration Sequence -

The Configuration Sequence is used to determine the state of all aspects of the subsystem and to change any of the default characteristics and status by host command.

The Configuration Sequence consists of the following steps:

1. The controller enters the 'controller online' state due to hard initialization. The host will receive a REINITIALIZED notice

from the controller.

2. The host issues a GET CHARACTERISTICS (UNIT=UN.CTR) command to the controller and waits for the END message.

3. The controller responds with an END message with current subsystem parameters. The parameters are those pertaining to the subsystem as a whole and include an indication of how many logical units are attached to the subsystem.

4. The host examines these characteristics, and if it desires to change any of them, it coordinates with any other host(s) attached to the controller, then issues a SET CHARACTERISTICS (UNIT=UN.CTR) command to the controller and waits for the END message. The controller will respond with an END message when the new characteristics are in effect. This step of the protocol is optional; if the host does not wish to change any of the characteristics, it does not execute this step.

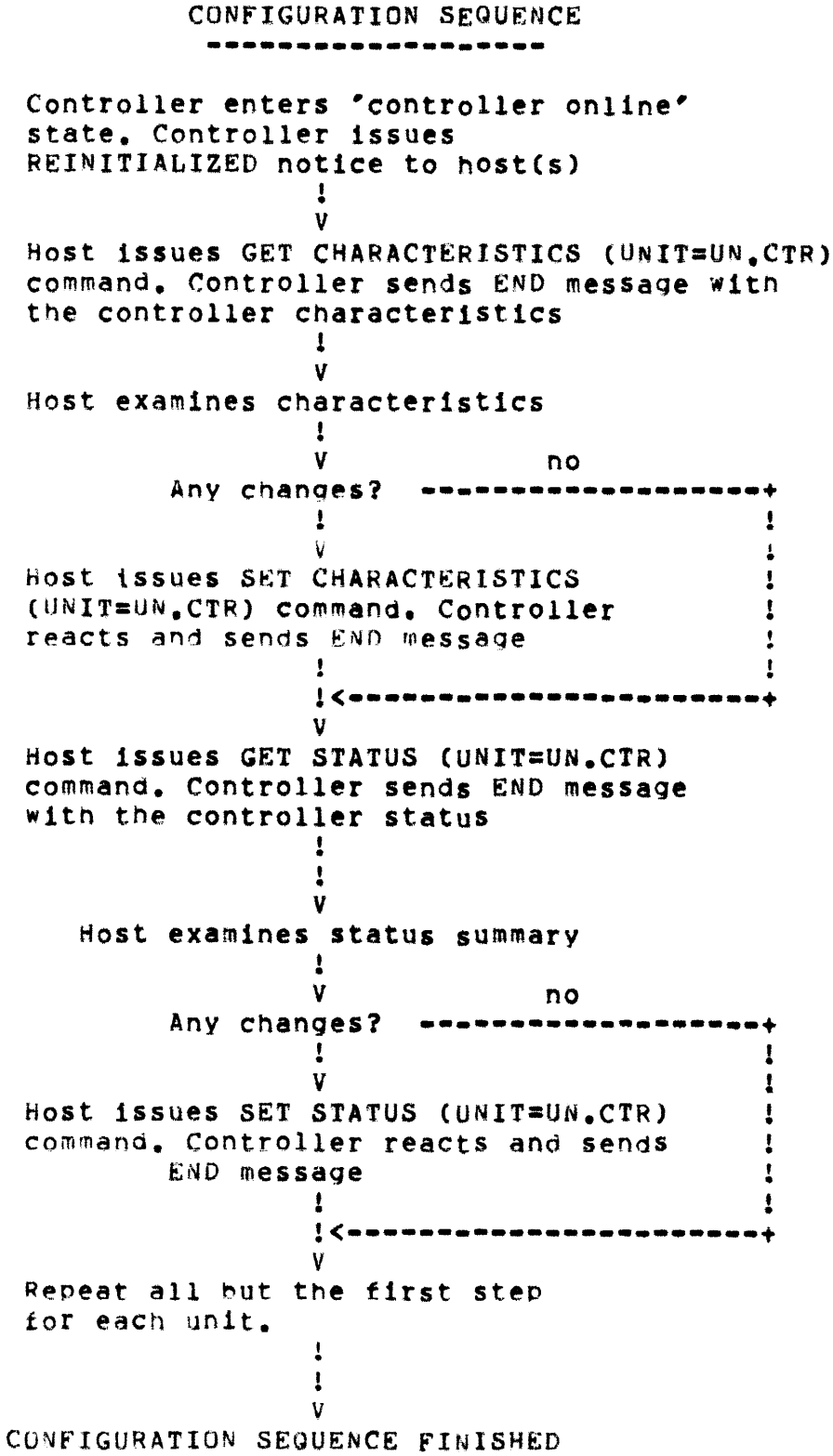
5. The host issues a GET STATUS (UNIT=UN.CTR) command to the controller and waits for the END message.

6. The controller responds with an END message with the current subsystem status. The status information contains status pertaining to the subsystem as a whole.

7. The host examines this status, and if it wishes to change the status of the controller, it issues a SET STATUS (UNIT=UN.CTR) command to the controller and waits for the END message. The controller will respond with an END message indicating that the desired status changes have taken effect. This step of the protocol is also optional; if the host does not wish to change the status of the controller, it does not execute this step.

After the above is completed, the host should do steps 2 thru 7 for each of the units attached to the controller which is of interest to the host.

A diagram of the configuration sequence follows:



4.5.2.5.2 Terminating Connections -

There are occasions when the controller must unilaterally break the logical connection with the host in an orderly fashion. This occurs when an controller that is online to the host must go offline due to a change in its host bus enable switches, an FE command removing the subsystem from service, or the controller determines it must reinitialize and is still sane enough to break the connection gracefully.

When the controller wishes to break its connection with the host, it simply sends a DISCONNECTING attention message to the host and ceases to accept further commands (they are refused with the appropriate error indication, if possible). Commands that were outstanding at the time the message is sent are allowed to complete, if possible.

If the host wishes to break its connection with the controller, it simply ceases to send further commands.

4.5.2.6 Interpretation of LOG, END, ATTENTION, and PSUEDO-TERM OUT messages -

The four types of messages sent by the controller to the host have the following broad interpretation:

1. END messages signal the completion (successful or unsuccessful) of a command. There is one and only one for each command received. These must of course be processed by the host because their reception stops the timing of the command and operates as a completion signal to the driver and thus to the operating system.

   There is summary unit status information in the END packet which is processed as appropriate by the system. The unit status information is most useful to determine which error recovery procedure to invoke for unsuccessful operations. More detailed utilization of the status information is a function of the sophistication and needs of the host driver. The more use is made of it in the END packet, the less GET STATUS operations should be necessary to recover from errors or maintain operating system tables.

2. LOG messages transmit information for the error log. It always contains information corresponding to an error condition in the subsystem, the severity of which corresponds to the class of the log message. LOG messages may or may not relate to a particular command, as indicated in the message itself. Systems may ignore log messages and the protocol will still function. Systems of any sophistication should make the log thresholds variable by the SET CHARACTERISTICS command, and should log the information in a meaningful way.

3. ATTENTION messages notify the host of significant changes in subsystem status or significant events internal to the subsystem that should be of interest to the host system. Attention messages never signify an error condition, although they may signify a status change that is the result of an error condition. For example, if a drive overheats, the host will potentially get two messages: a LOG message indicating the overheating condition, and an ATTENTION message indicating that the drive is no longer available to the host. Other examples of ATTENTION messages are notification that a particular deteriorating sector on a disk warrants replacement or that the subsystem is changing status to WRITE PROTECT.

   In order for the connection protocol to work, REINITIALIZATION attention messages must be at least minimally processed by the host. The interface will still function properly if other attention messages are ignored by the host. These other attention messages give sophisticated systems the capability to react dynamically to status changes within the subsystem.

4. PSUEDO-TERM OUT messages provide a mechanism for sending text to be displayed at a host terminal which is being used as the HSC50 FE panel. The use of a psuedo-terminal must be explicitly enabled by the host. The PSUEDO-TERM IN and PSUEDO-TERM OUT commands are used to transfer ASCII strings between the host and the controller. The communication is asynchronous. While enabled the messages cannot be ignored by the host.

### 4.5.3 Shadowing

#### 4.5.3.1 General Rules Governing Use of Shadowing -

A shadow relationship may be established between any two disk units. A disk unit may have only one shadow relationship. Any WRITE to a unit with a shadow relationship will cause the identical WRITE on the other unit.

At the time the shadow relationship is established, the host may specify that the first unit is to be copied onto the second as a background activity while the units are in use. Since the copy may not be performed sequentially, if an unrecoverable error occurs before the copy is complete, the state of the second unit is undefined. When the copy is completed, an ATTENTION message is sent to the host.

A WRITE operation to a shadowed unit on a cached system must be write-through. A write-back operation to a shadowed unit will be

rejected.

#### 4.5.3.1.1 Restricted Command Subset Available on Shadowed Units. -

Because shadowed disk units in a shadow relationship must be managed by the controller, there are certain commands that are not legal to units of this type.

The following commands are illegal, except for breaking the shadow relationship, to a disk unit that is currently shadowing another disk unit:

SET CHARACTERISTICS
SET STATUS

A WRITE with COMPARE to a shadowed unit will cause the compare to be performed for both units.

### 4.5.3.2 Establishing and Terminating Shadow Relationships -

Shadowing is a symmetrical relationship limited to two units of the disk class type. Any WRITE operation to either unit is reflected on the other. Shadow relationships will be controller level characteristics, established and terminated via the SET CHARACTERISTICS (UNIT = UN.CTR) command.

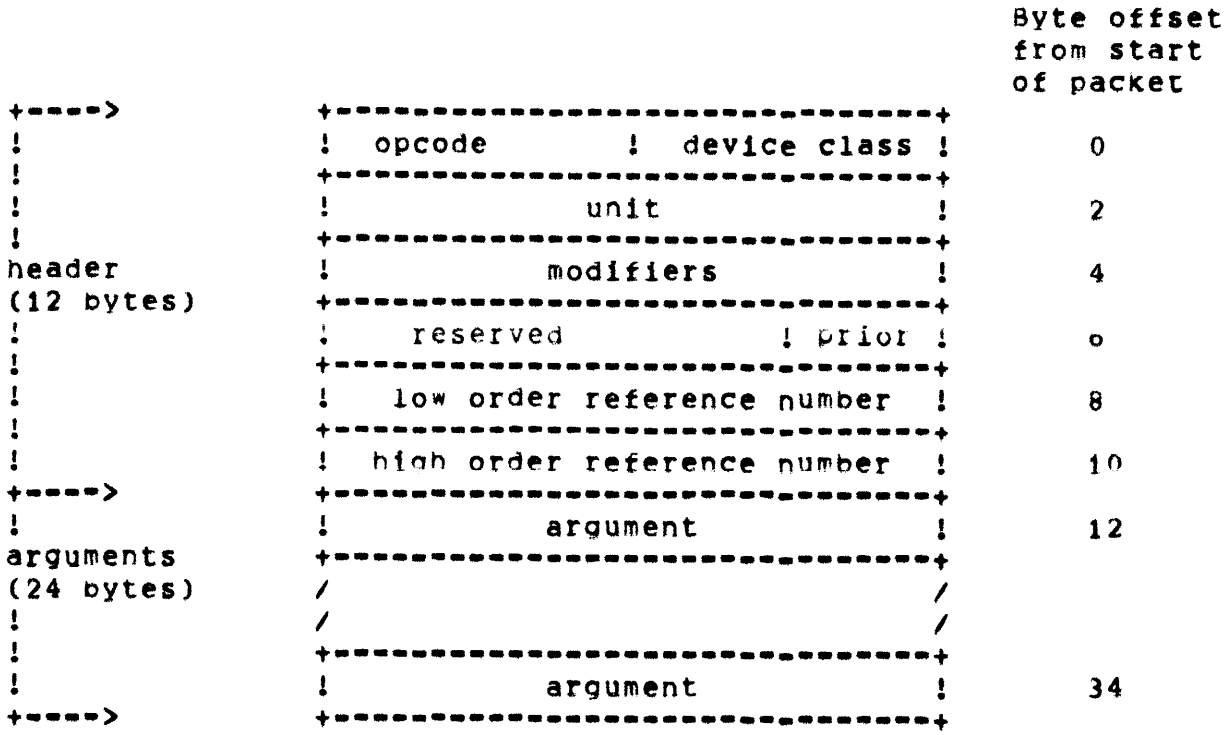
### 4.5.3.3 Behavior of Shadow Requests Under Error Conditions -

The END packet will not be sent for a shadowed WRITE until the the operation is completed on both units. If there is an unrecoverable error on one unit, e.g. a head crash, the controller will still attempt to complete the WRITE on the other unit. The host should then break the shadow relationship to use the operational disk.

### 4.5.4 General Command Packet Format

All packets consists of an 12 byte packet header and a 24 byte argument area. Arguments are always multiples of two bytes. Multiple-byte arguments are in VAX-11 format.

Command packets from the host to the controller are in the following general format:




The PRIORITY specifies the priority for the command. The HSC50 will accept two priority levels -- normal: PR.NRM and high: PR.HI.

The OPCODE field specifies the command. An opcode of 255(10) is interpreted as ESCAPE. In this case, the actual opcode will be taken from a \to be specified\ argument word.

The device class field specifies the device class of the unit to which the command is directed. The HSC50 will accept three device classes -- controller: DV.MSC, disk: DV.DSK and tape: DV.TAP. All references to the cache are implicit through one of the other device classes.

The UNIT number specifies the unit within the subsystem that the command refers to. Both the DEVICE CLASS and the UNIT are necessary to uniquely identify a specific unit.

Unit UN.CTR is the controller itself (and sometimes the entire subsytem

by implication).

The REFERENCE NUMBER is an "ID" for the command. Note that it MUST BE UNIQUE for the outstanding commands of that host and it MUST NOT BE ZERO.

The MODIFIERS specify variations to a command.

There are several arguments which appear in a number of commands. They are:

1. Byte Count -- specifies the number of bytes to be transferred

2. Buffer Descriptor -- specifies the buffer in the host memory, always paired with a host port id

3. Host Port ID -- specifies the port id of the host to/from which data is to be transferred

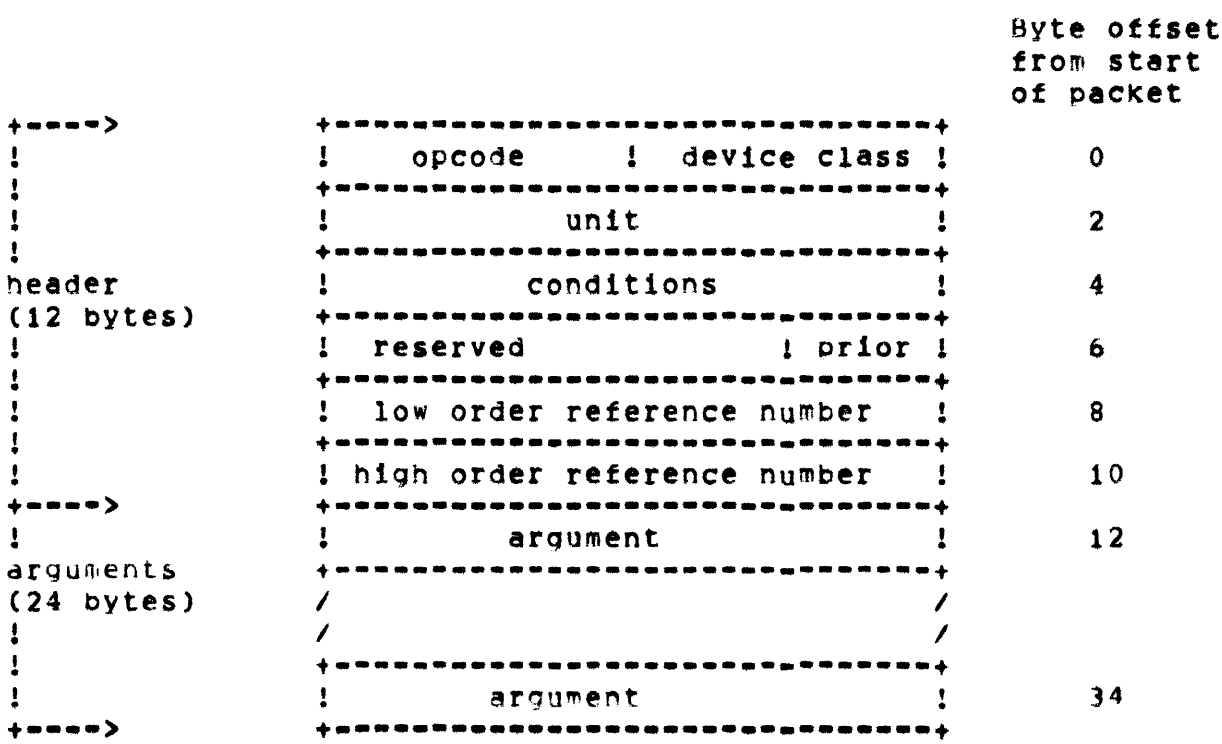
4. Block Number (LBN) -- specifies the logical block number in the disk unit's storage

5. Block Size -- specifies the block (physical record) size for tape class devices

6. Block Count -- specifies the number of blocks for tape class devices

7. Reserved -- a field reserved for future use, MUST BE ZERO.

Message packets from the controller to the host are in the following general format:




The PRIORITY,OPCODE,DEVICE CLASS,UNIT and REFERENCE NUMBER are as described for host-to-controller packets.

The CONDITIONS specify command execution results, especially with regard to errors.

### 4.5.5 Host-to-Controller Commands

This section describes the individual commands the host can issue to a controller. The numerical values for opcodes, modifiers, and other bit-specific information ( such as the format of status and characteristics blocks) are in the tables at the end of the section.

The discussion of each command includes a statement describing the "gatekeeper class" of the command. This refers to the relationship that the command has to any other commands that are outstanding at the time the command arrives and any command that might follow it while it is still outstanding. The "gatekeeper" is a function internal to the controller that decides when processing for a given command can begin. Commands that cannot be processed immediately are queued at the gate, and are allowed to enter the subsystem by the gatekeeper when appropriate. Note that the gatekeeper does not impact whether or not a command is accepted; all valid commands are accepted immediately. Rather, the gatekeeper forces serial processing on certain commands, and its only visibility on the part of the host is the increased latencies for commands blocked at the gate. The gatekeeper is required to provide the LBN FIFO guarantee to the host, and to assure consistency of operation across classes of commands that affect subsystem parameters.

There are five gatekeeper classes:

1. GET STATUS class. GET STATUS class commands have no impact on any activity in the subsystem and always begin processing immediately unless they are blocked at the gate by a SET STATUS class command. GET STATUS class commands do not block other commands.

2. READ class. A READ class command is allowed to enter the subsystem immediately and begin processing as long as its entry is not blocked by a WRITE class or SET STATUS class command that preceeded it. READ class commands do not block any commands that may follow it.

3. WRITE class. A WRITE class command is allowed to enter the subsystem and begin processing only if no other commands to the same LBN(s) are already in the system, and the gate is not closed by a SET STATUS class command. If another command involving one or more of the same LBN(s) is already in the subsystem, the WRITE class command is queued at the gate, and does not begin processing until all previous commands to the affected LBNs complete. WRITE class commands will block any commands that follow it that involve one or more of the same LBNs until the WRITE class command itself has completed.

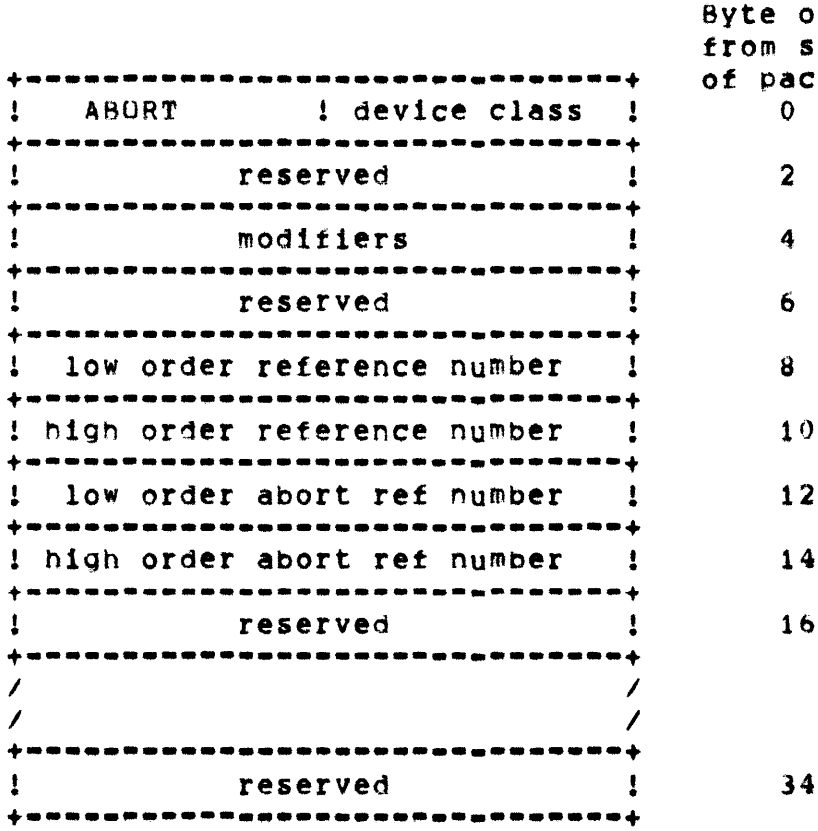
4. SET STATUS class. SET STATUS class commands affect the operation of a unit or the subsystem itself. Therefore, they do not enter the subsystem until all outstanding operations to the affected unit(s) are complete, and any operations to the

affected unit(s) that follow it are blocked at the gate until the SET STATUS class command itself completes. If the SET STATUS class command operates on the controller, all outstanding requests are completed before it is begun, and it is the only command allowed to be active until it completes.

5. Immediate Execution class. INIT and ABORT are Immediate Execution class commands and are not delayed for any reason by the gate keeper.

4.5.5.1 ABORT -

Format:




Allowable Units:

Must be the same as in the command to be aborted.

Arguments:

Reference number of the command to be aborted.

Function and Results:

The specified command is aborted at the earliest point convenient for the controller. If any-unit affecting action has already been taken, an END message for the original command is sent to the host indicating the the degree of command execution(e.g., number of bytes transferred.) If no action has yet been taken, the END message will so indicate. After the abort action has been completed, an END message for the ABORT command is sent to the host.

When the END for the ABORT is received by the host, the original command is no longer active in the subsystem.

Gatekeeper Class:
---

ABORT is an immediate execution class command like INIT and bypasses the gate keeper.

ABORT END Packet:
---

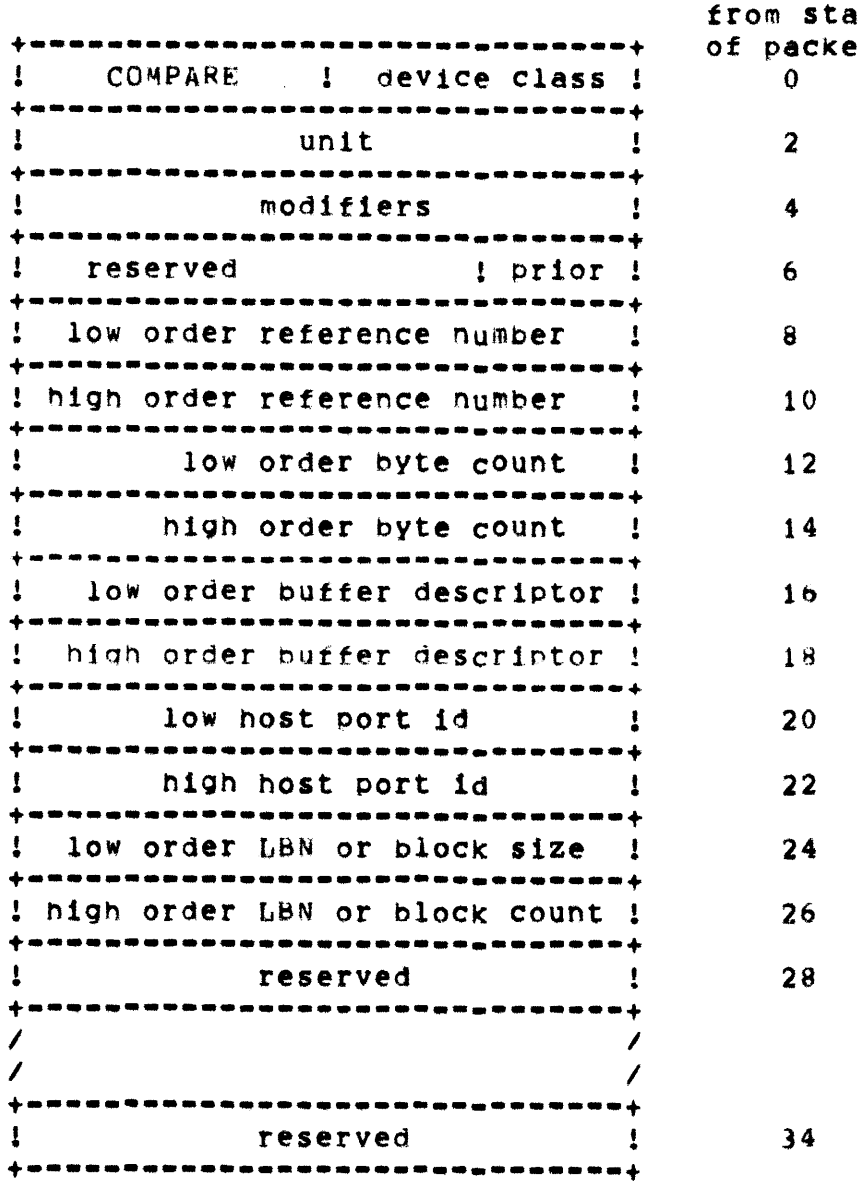
Allowable units:
---

Same as in the original command packet.

Conditions:
---
Error - abort reference number not a current outstanding reference number.

4.5.5.2 COMPARE -

Format:




Allowable Units:

Any other than UN.CTR.

Arguments:

Byte Count
Buffer Descriptor
Block Number

Function and Results:
---

COMPARE directs the controller to compare a specified number of bytes from the specified unit against host memory. Although the data compared is contiguously organized, the comparison itself may not occur contiguously. The compare terminates when the byte count is satisfied or an error occurs.

Gatekeeper Class:
---

COMPARE is a READ class command.

Special Comments:
---

If the LBN being compared happens to reside in the cache, the compare operation results in a FLUSH of the cache data, invalidation of the cache entry, and a read of the data from the disk without a reload into the cache.

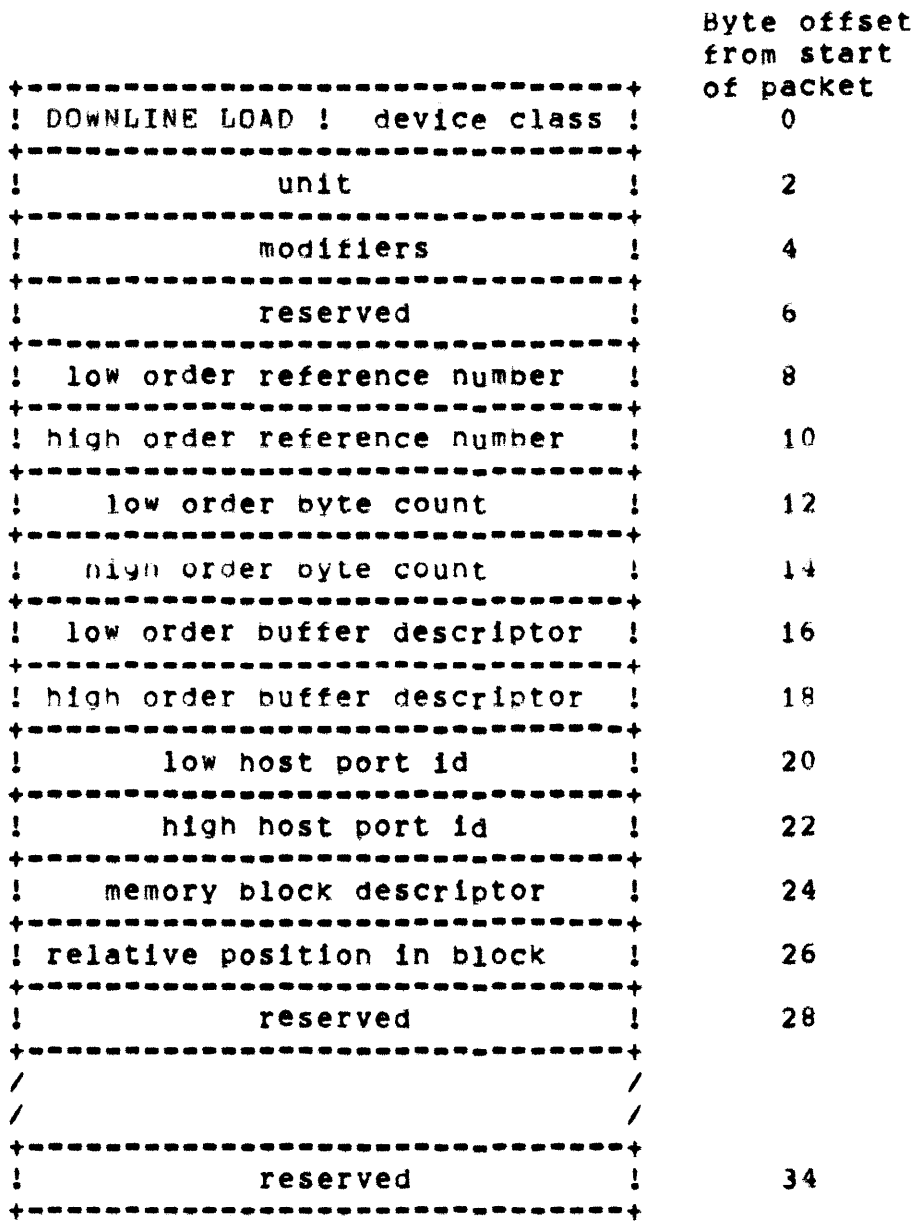
COMPARE END Packet:
---

Conditions:
---

Error - Compare failure, relative byte number where compare failed follows the last argument of the normal END packet.

4.5.5.3 DOWNLINE LOAD -
(Only on the UDA50, not implemented on the HSC50)

Format:




Allowable units:

Any unit in the UDA50 subsystem which is capable of accepting dowline loading of code.

Arguments:

Byte Count
Buffer Descriptor
Memory Descriptor

Modifiers:
---

None

Function and results:
---

The contents of the host buffer are loaded into the unit's memory beginning at the relative position in the specified memory block.

\The format of the downline load information is to be specified in a document by Bill Grace. The information must include the specification of each of the rtn descriptors which may be used in a subsequent EXECUTE command.\

Gatekeeper Class:
---

DOWNLINE LOAD is a SET STATUS class command.

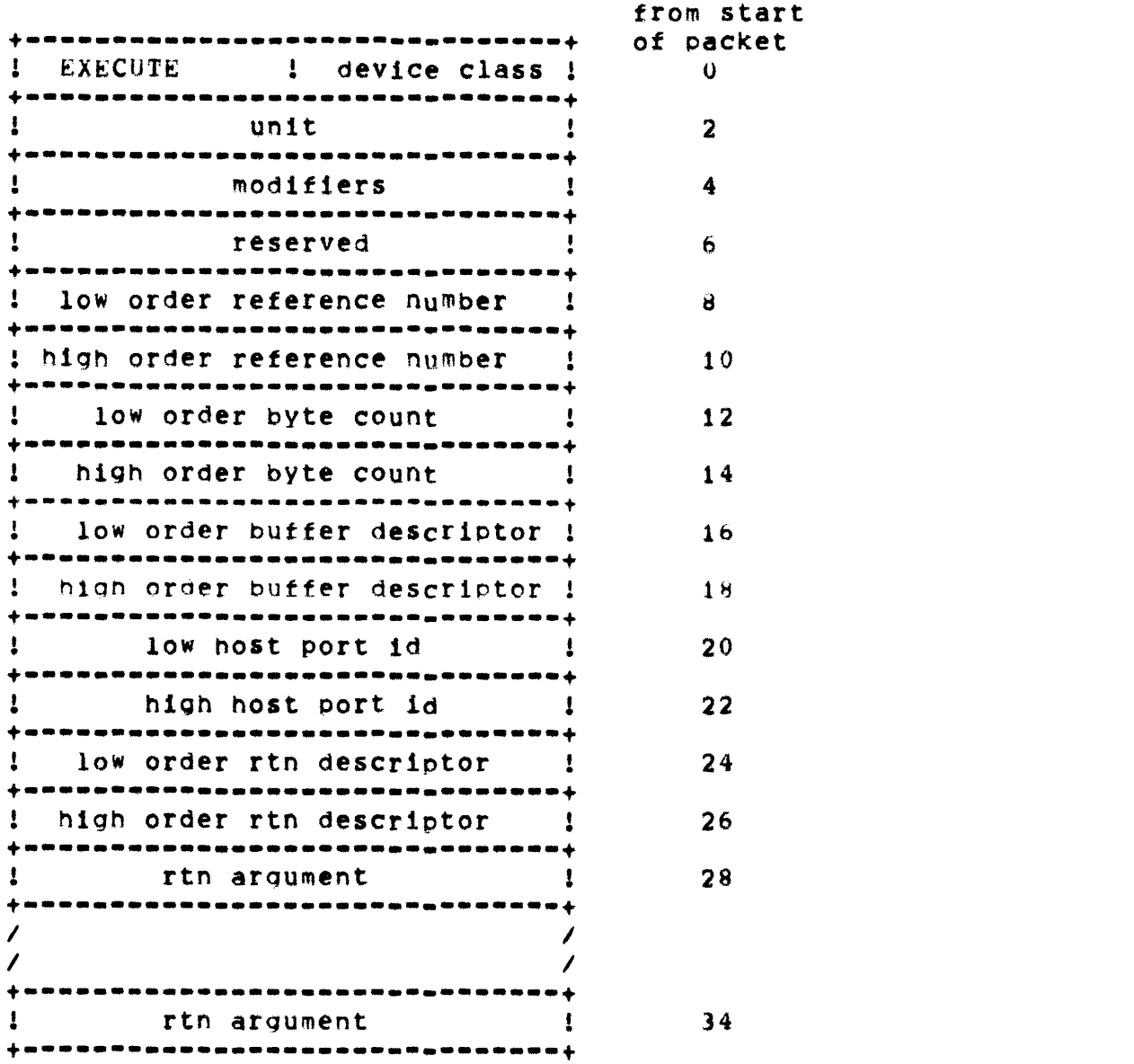
DOWNLINE LOAD END Packet:
---

Defined conditions:
---

Error - Space available in the unit is inadequate for the routine.
Error - Another routine is in execution on the unit.

4.5.5.4 EXECUTE -

Format:




Allowable Units:

UN.CTR only on the HSC50. On the UDA50, any unit that has the capability.

Arguments:

Byte Count
Buffer Descriptor
Routine Descriptor
Routine Arguments

Function and Results:
---

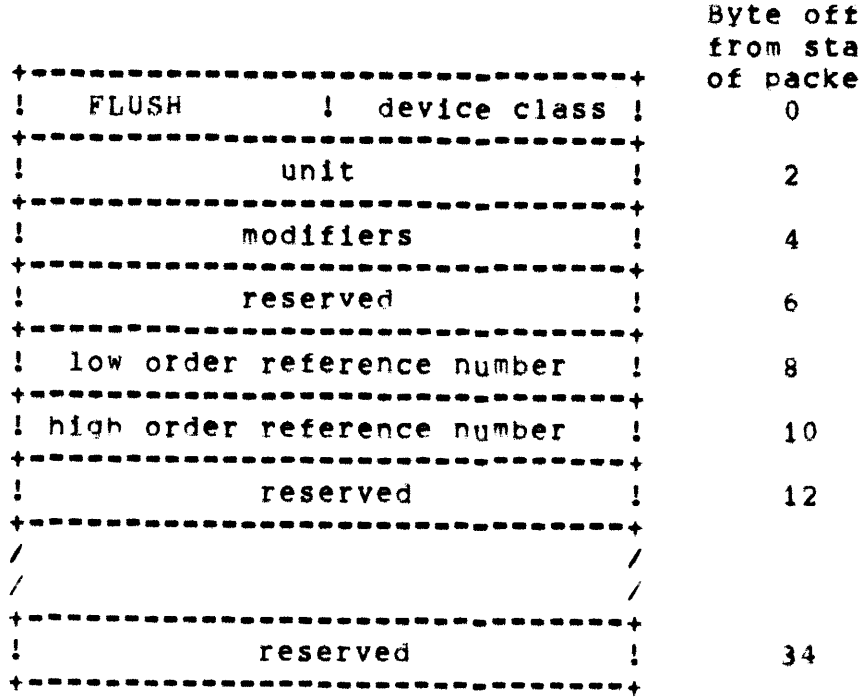
The specified routine is executed and the results returned to the host buffer. If the buffer descriptor is zero the results are returned in the END message.

Gatekeeper Class:
---
EXECUTE END Packet:
---

Defined conditions:
---
Error - routine does not exist Error - argument list wrong size Error - a routine is in execution on the unit

4.5.5.5 FLUSH -

Format:




Allowable units:

UN.CTR or any disk unit. UN.CTR causes the entire cache to be flushed. Any other unit causes only those entries for the specified unit to be flushed.

Arguments:

None.

Function and results:

A FLUSH command directs the controller to search its cache tables for dirty entries that correspond to the specified unit, and if any such entries are found, to WRITE them out to the disk. The END packet is sent when all dirty entries have been written.

After flushing any dirty data from the cache, the controller leaves the entry in the cache, but marks it as clean.

If no dirty entries corresponding to the specified unit are found, this command is effectively a very slow NOP.

Gatekeeper Class:
---

FLUSH is a SET STATUS class command.

Special Comments:
---

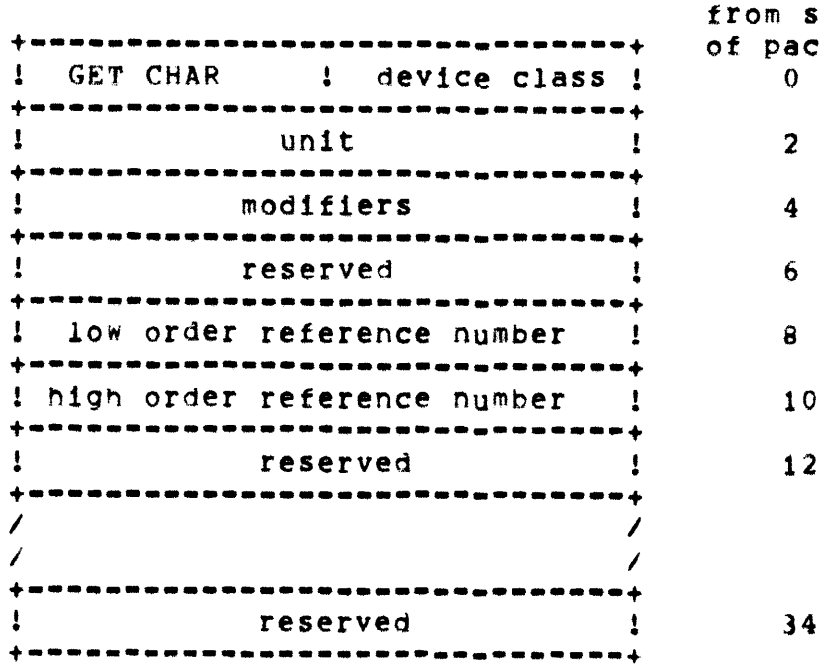
FLUSH is primarily useful as a means of "committing" temporary or tentative data to the disk in a write-back environment.

FLUSH END Packet:
---

Defined conditions:
---

4.5.5.6 GET CHARACTERISTICS -

Format:




Allowable Units:

Any.

Arguments:

None.

Function and Results:

The specified unit's characteristics are returned in the END packet. A definition of the characteristics format can be found in Section 4.5.7.

Gatekeeper Class:

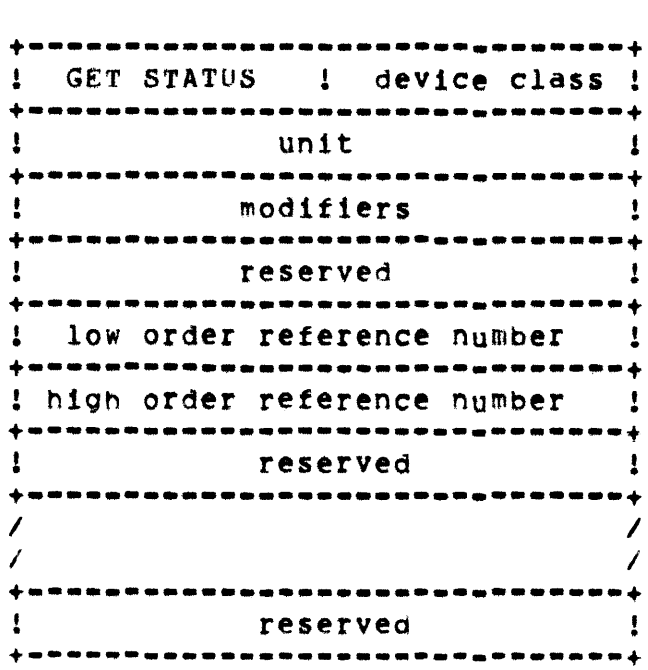
GET CHARACTERISTICS is a GET STATUS class command.

GET CHARACTERISTICS END Packet:

Defined Conditions:
---

4.5.5.7 GET STATUS -

Format:




Allowable Units:

Any.

Arguments:

None.

Function and Results:

The status of the specified unit is returned in the END packet. The format of the status returned is given in Section 4.5.7.

Gatekeeper Class:

GET STATUS is of course a GET STATUS class command.

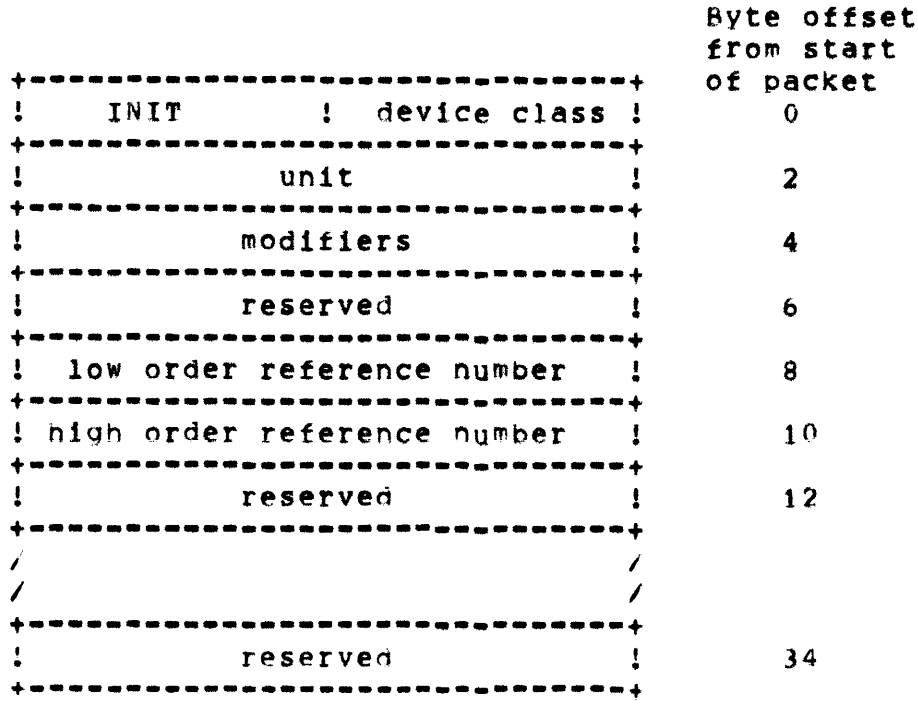
Special Comments:

GET STATUS END Packet:

Defined Conditions:
---

4.5.5.8 INIT -

Format:




Allowable Units:

UN.CTR only.

Allowable Modifiers:

None

Arguments:

None.

Function and Results:

An INIT command is used as described in the resynchronization section to resynchronize with the controller. It causes the controller to cease processing on any outstanding host commands, and to reinitialize all internal context relative to the initializing host. The controller does not stop running; it merely clears the controller of all commands relative to the issuing host and returns to default characteristics and

status for that host only. The other host (if any) is not affected. Any outstanding requests that are successfully stopped by the controller are not responded to; they are "lost". The controller will respond with an END packet when the soft initailization is complete (all outstanding operations are cleaned up and internal context is in an initial state).

Note that this INIT function is distinct from and much less drastic than a bus-level hard init.

Gatekeeper Class:
---

Because INIT must begin its work immediately, it is a special case command. It is never blocked at the gate.

Special Comments:
---

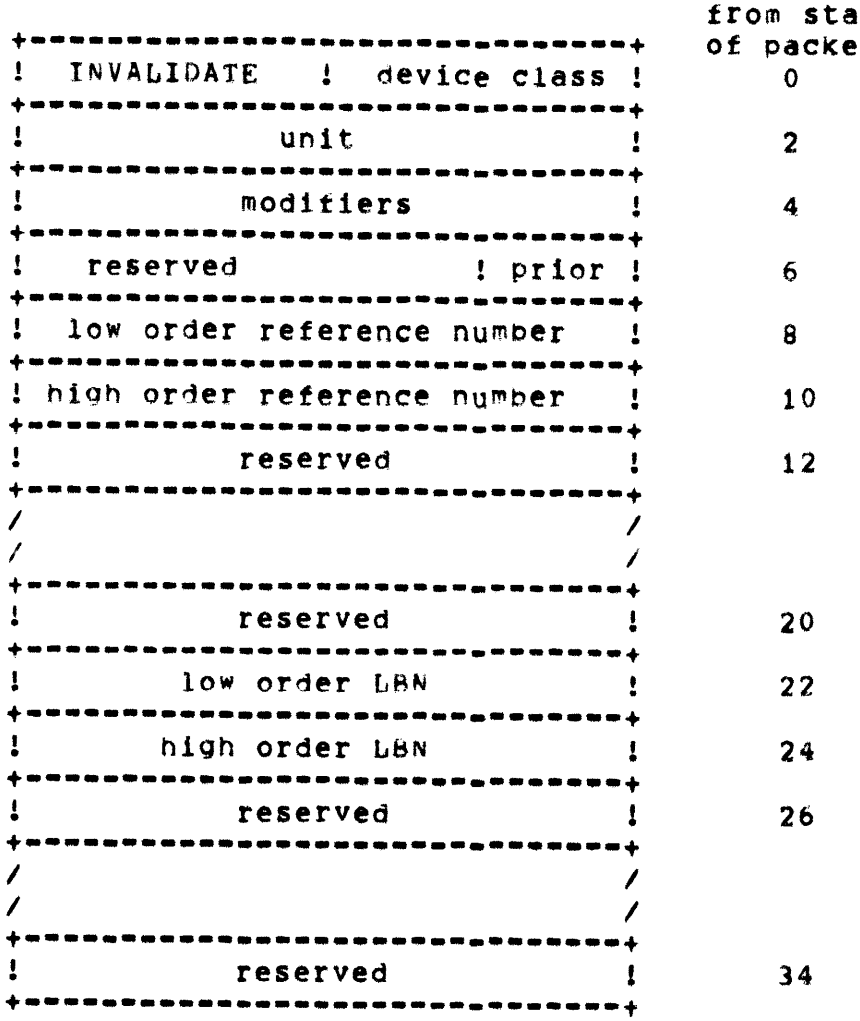
When the controller receives an INIT command, it attempts to stop the processing of any host requests that are already in progress. Because of parallelism within the controller, however, it is impossible to stop processing on requests that have passed a certain internal processing point. Therefore, the host should be ready to process END packets corrresponding to any outstanding requests even after a INIT command has been sent. The host should remain ready to process END packets for outstanding commands until the END packet for the INIT itself is received, at which point the host can be assured that no further END packets will be forthcoming.

INIT END Packet:
---

Defined conditions:
---

4.5.5.9 INVALIDATE -

Format:




Allowable units:

Any disk class unit. The unit qualifies the LBN argument to indentify the unique cache LBN to be invalidated.

Arguments:

The block number specifies which LBN on the specified unit is to be invalidated.

Function and results:

This command directs the controller to delete the specified LBN from the cache directory without a write-back of the dirty data.

This command is a limited-use command provided to allow the host to recover from errors resulting from failure of a CCD block that contains "dirty" write-back data; it eliminates the error that occurs on reads from "bad" dirty cache blocks. Its main use would occur when a temporary file is deleted after a cache block failure. The disk LBN would now become available for normal operations, while the "bad" cache block(s) remain out of service.

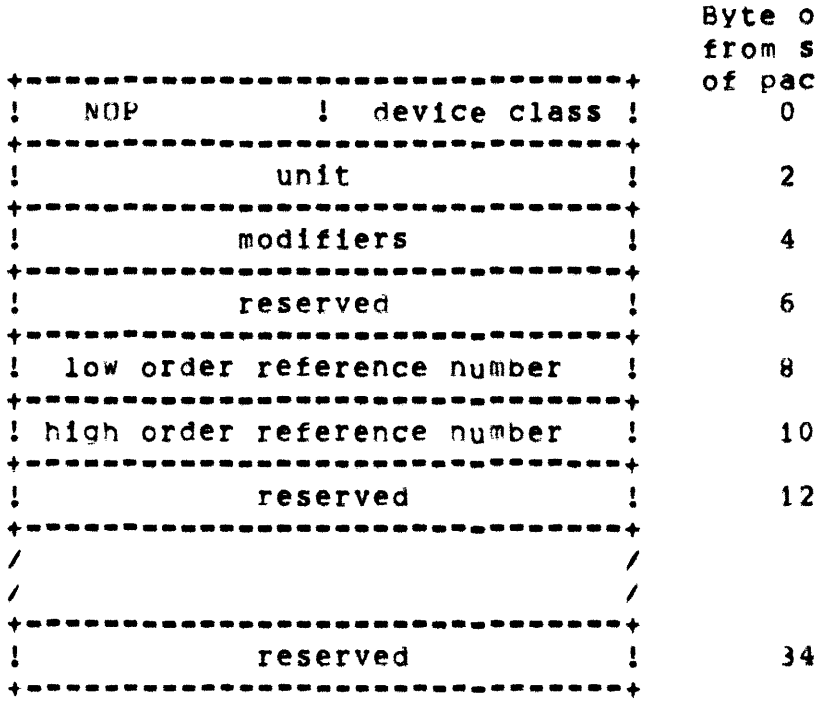
Gatekeeper Class:
---

INVALIDATE is a WRITE class command.
INVALIDATE END Packet:
---

Defined conditions:
---

4.5.5.10 NOP -

Format:




Allowable Units:

UN.CTR only.

Arguments:

None.

Function and Results:

This command causes the controller to respond with an END packet without performing any internal operations other than that necessary to respond. Like all END packets, the response to this command contains summary status information for the specified unit.

Gatekeeper Class:

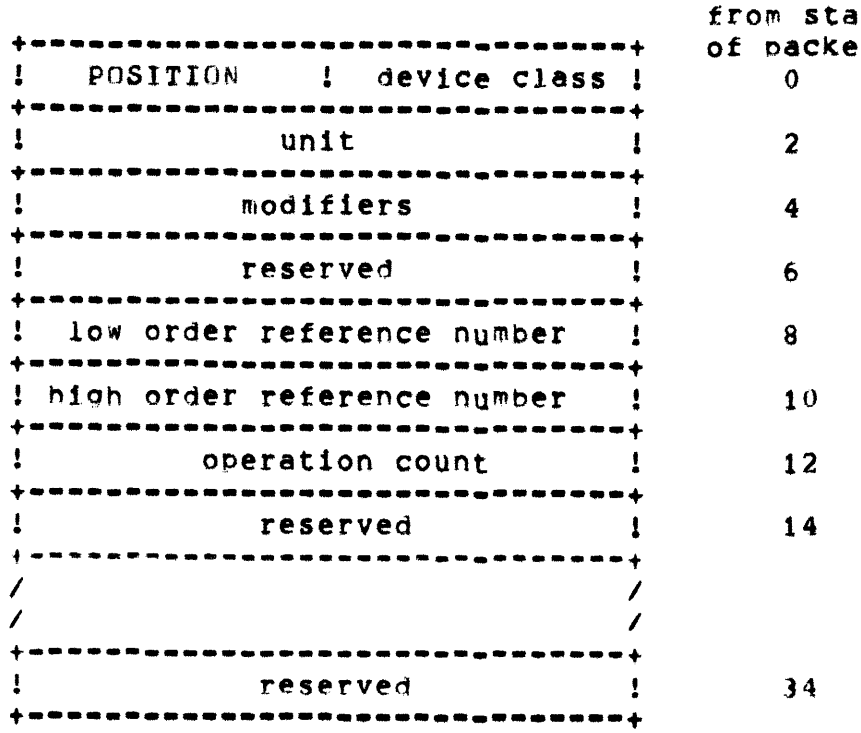
NOP is a GET STATUS class command.

NOP END Packet:

Defined conditions:

4.5.5.11 POSITION -

Format:




Allowable Units:

Any tape class unit.

Arguments:

Operation Count

Allowable modifiers:

1. PO.FIL -- POSITION to file.

2. PO.BOT -- POSITION to beginning of tape (REWIND).

3. PO.UNL -- POSITION and unload. This modifier is used only in conjunction with PO.BOT and causes an unload after the rewind.

4. PO.LEI -- POSITION to logical end-of-tape.

Function and Results:

If there are no modifiers, the tape is positioned relative to blocks. If PO.FIL is set, the tape is positioned relative to file marks. In these two cases, the operation count specifies the number of positions and the sign specifies the direction (forward if positive, backward if negative.) If PO.BOT (and optionally PO.UNL) or PO.LET is set, the specified positioning occurs and the operation count is ignored.

Gatekeeper Class:
---

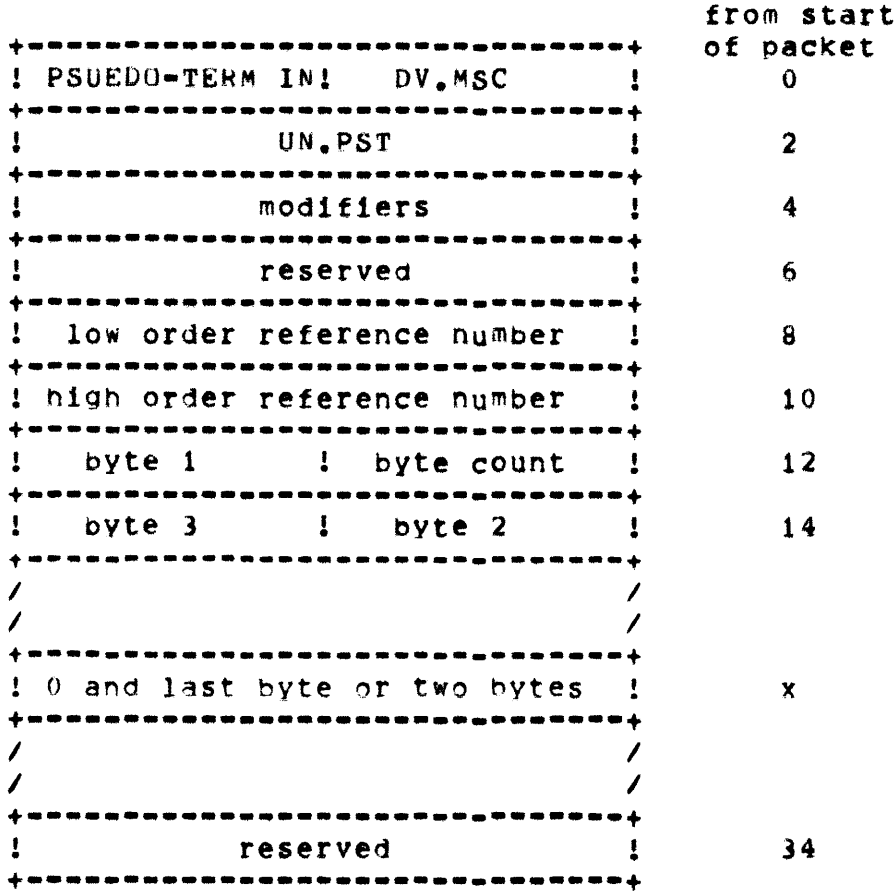
POSITION is a SET STATUS class command.

POSITION END Packet:
---

Defined conditions:
---

4.5.5.12 PSUEDO TERM IN -

Format:




Allowable units:

UN.CTR.

Arguments:

The byte count is the length of the ASCII string which follows it in the command packet.

Function and results:

The PSUEDO-TERM IN command directs the controller to obtain the ASCII string from the command packet, and to process the string therein as if that information was entered at the HSC50 FE panel. When the input string has been successfully accepted , the END packet is returned to the host.

PSUEDO-TERM IN commands are not required for normal operation since the functions can be invoked at the FE panel.

Gatekeeper Class:
---

PSUEDO TERMINAL is a SET STATUS class command.

Special Comments:
---
The PSUEDO-TERM IN and PSUEDO-TERM OUT messages serve to provide an alternate path for conversing with the HSC50 controller via its terminal interface. The interactions that occur across the psuedo-terminal are identical to those that occur on a physical terminal attached to the controller; the difference lies in the mechanism (ICCS vs ASCII line), and the fact that a program as well as a human can interact with the psuedo-terminal.

It is expected that one mode of host utilization will be a "pass-through" application program that accepts and prints psuedo-terminal strings on a host user terminal, making that terminal appear as if it were attached directly to the HSC50.

The PSUEDO-TERM IN command exists to allow any operation that can be performed at the HSC50 FE panel to be performed from the host as well.

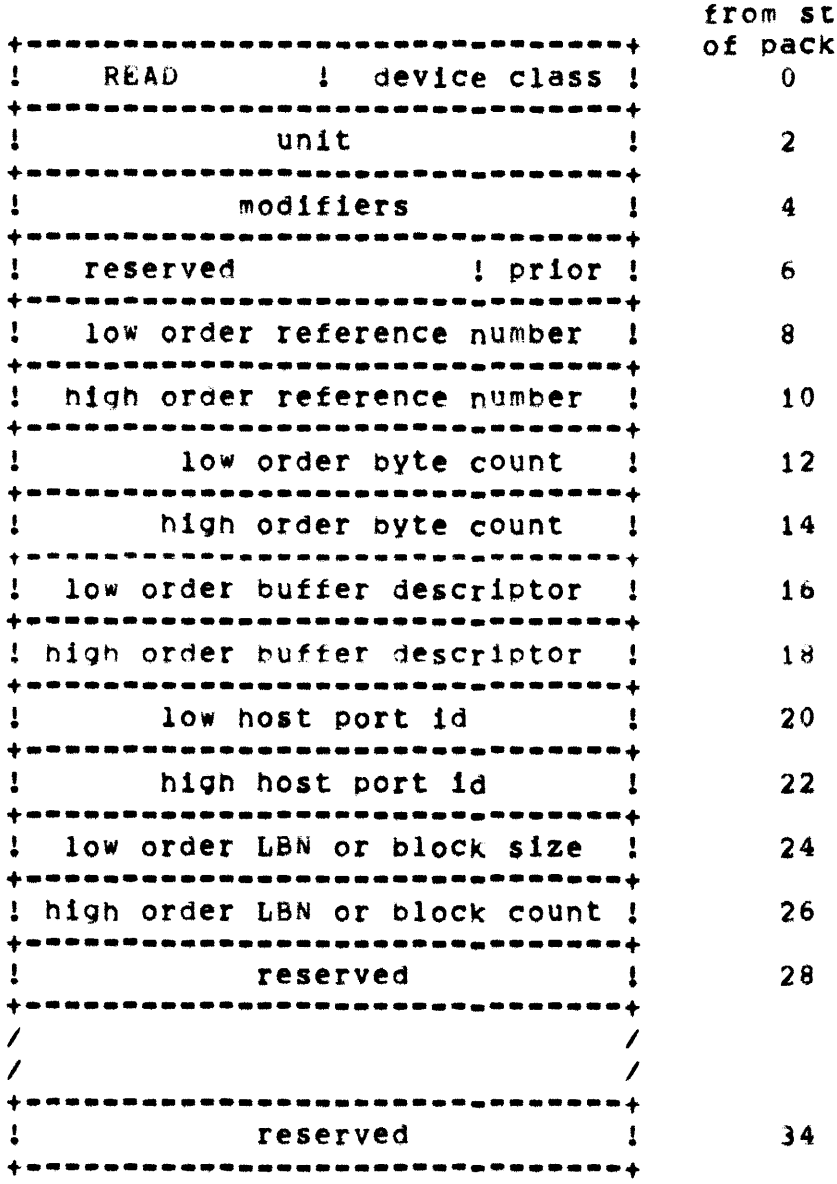
\The syntax and semantics of all possible FE panel interactions must be defined for the HSC50. This set will be valid for the PSUEDO-TERM IN command.\

PSUEDO-TERM IN END Packet:
---

Defined conditions:
---

4.5.5.13 READ -

Format:




Allowable Units:

Any unit except UN.CTR.

Arguments:

Byte Count
Buffer Descriptor
Block Number

Block Count
Block Size

Allowable modifiers:
---

1. RD,NEC -- READ no ECC correction. Causes the READ to return the data without any ECC correction applied.

2. RD,REV -- READ reverse. Used on tape units only to READ in the reverse direction.

3. RD,CMP -- READ with compare. Causes the READ to be followed by a COMPARE after the read data has been shipped to the host memory.

4. RD,NOC -- READ no cache. Causes a cached system to FLUSH the specified LBN if it is in the cache and dirty, invalidate the cache entry, and read the data from the disk without loading it into the cache. If the LBN is not in the cache, this modifier causes a disk read without loading the block into the cache.

Function and Results:
---

READ directs the controller to transfer the specified number of bytes from the specified unit to host memory. The contents of the logically contiguous device locations which begin with the first byte of the specified block are transferred to the virtually contiguous locations in host memory beginning at the first byte location specified by the buffer descriptor. Although the data transferred is contiguously organized, the transfer itself need not occur sequentially. The transfer terminates when the byte count is satisfied or an unrecoverable device error occurs. An END message is then sent containing summary status information.

For a tape class device, the block specifies the expected number of bytes in each block and the block count specifies the expected number of blocks required to satisfy the byte count. If the block count is negative, the read is in the reverse direction.

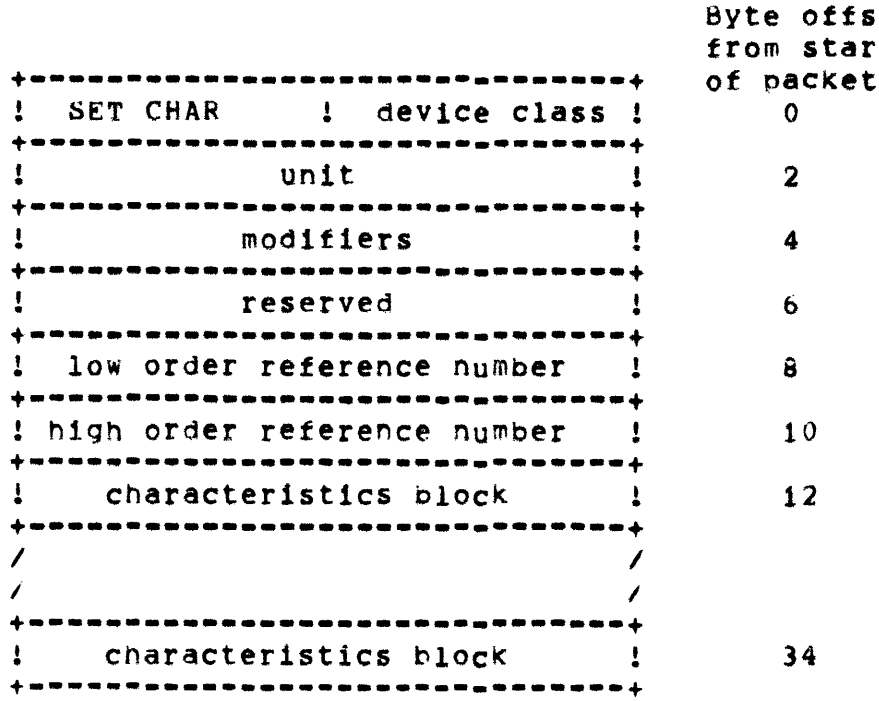
Gatekeeper Class:
---

READ is of course a READ CLASS command.
READ END Packet:
---

Defined conditions:
---

4.5.5.14 SET CHARACTERISTICS -

Format:




Allowable Units:

Any that has settable characteristics.

Arguments:

Characteristic Block

Function and Results:

Each of the specified characteristics is set to the specified value. A description of the characteristics is found in Section 4.5.7.

Gatekeeper Class:

SET CHARACTERISTICS is a SET STATUS class command.

Special Comments:

The controller will honor changes for characteristics changes from any

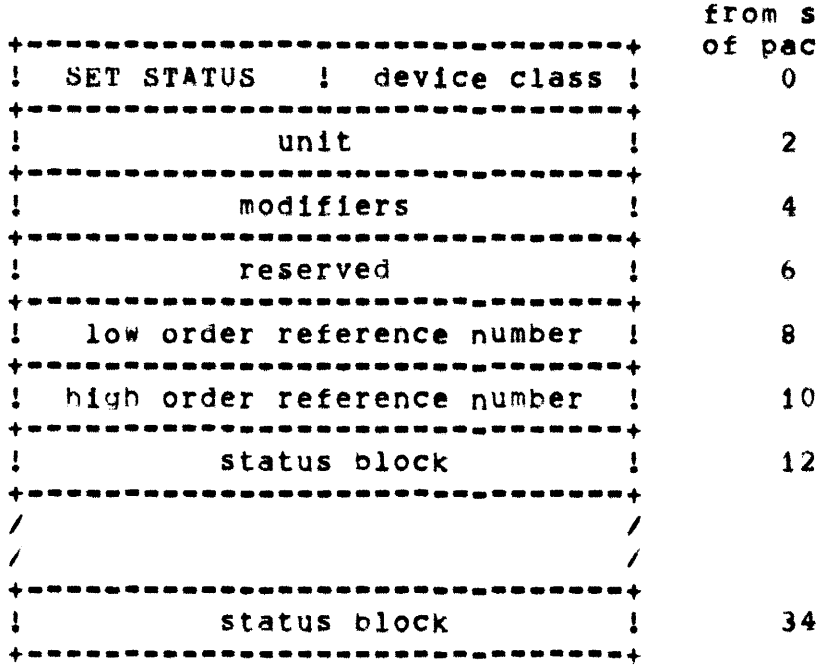
host. The hosts in a multiple-host environment are responsible for coordinating characteristics changes so that they do not begin working at cross purposes.

SET CHARACTERISTICS END Packet:
---

Defined conditions:
---

4.5.5.15 SET STATUS -

Format:




Allowable Units:

Any other than a disk unit shadowing another disk unit.

Arguments:

Status Block

Function and Results:

Each specified status is set to the specified value. The description of the status format can be found in Section 4.5.7.

Gatekeeper Class:

SET STATUS is of course a SET STATUS class command.

Special Comments:

The controller will honor requests for status changes from any host.

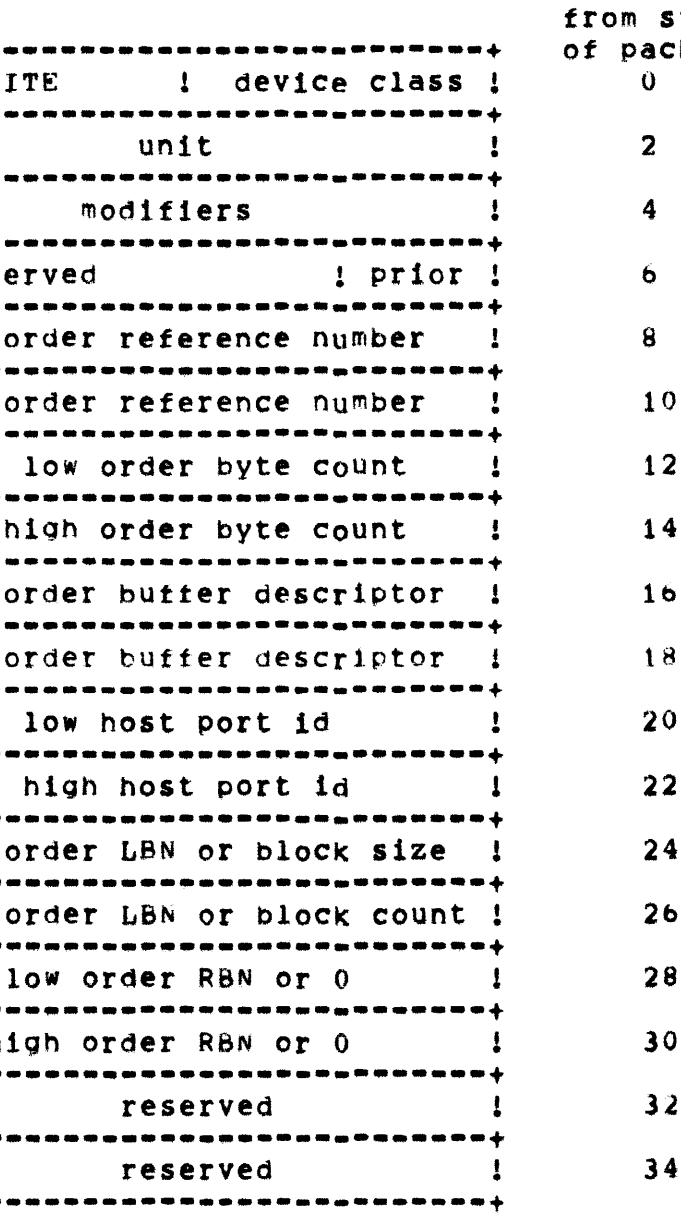
The hosts are responsible for coordinating status changes to units they are sharing.

SET STATUS END Packet:
---

Defined Conditions:
---

4.5.5.16 WRITE -

Format:




Allowable Units:

Any unit other than UN.CTR.

Arguments:

Byte Count
Buffer Descriptor
Block Number
Replacement Block Number
Block Size
Block Count

Allowable modifiers:
---

1. WR.CMP -- WRITE with compare. Causes the WRITE to be followed by a COMPARE after the data has been written on the disk.

2. WR.BCK -- WRITE back. Causes a cached system to load the data into the cache as "dirty" data. The default is write-through.

3. WR.NOC -- WRITE no cache. Causes a cached system to invalidate any cache entry corresponding to this LBN and then write the data to the disk.

4. WR.NOS -- WRITE no shadow. Causes the controller to avoid shadowing this write when the target unit is shadowed. If the target unit is not shadowed, this modifier is ignored.

5. WR.REP -- WRITE with replacement. Causes the susbsystem to replace the bad LBN with a replacement block and write the data into the replacement block. This modifier limits the byte count to one sector. The RBN must be zero for the HSC50 since it determines the actual RBN.

Function and Results:
---

WRITE directs the controller to transfer the specified number of bytes from host memory to the specified unit. The contents of the virtually contiguous memory locations which form the host buffer are transferred to the logically contiguous device locations which begin at the first byte of the specified block. Although the data is contiguously organized the transfer itself need not occur sequentially. The transfer terminates when the byte count is satisfied or an unrecoverable device error occurs. An END message is then sent containing summary status and the number of bytes successfully transferred.

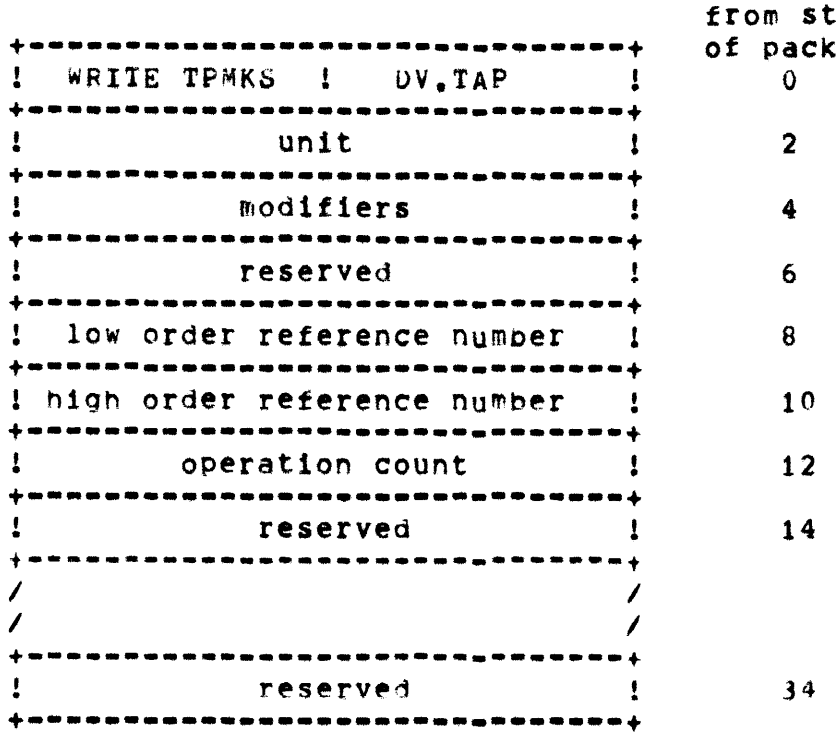
Gatekeeper Class:
---

WRITE is of course a WRITE class command.
WRITE END Packet:
---

Defined conditions:
---

4.5.5.17 WRITE TAPE MARKS -

Format:




Allowable Units:

Any tape class unit.

Arguments:

Operation Count

Allowable modifiers:

1. WT.MRK -- WRITE TAPE mark. Causes the subsystem to write a tape mark on the specified unit.

2. WT.EGP -- WRITE extended gap. Causes the controller to write an extended gap on the specified tape unit.

3. WT.DSE -- Data Security Erase tape.

4. WT.LET -- WRITE Logical End of Tape.

Function and Results:
---

The operation specified by the modifiers is performed the number of times specified by the operation count.

Gatekeeper Status:
---

WRITE TPMRKS is a SET STATUS class command.

WRITE TPMRKS END Packet:
---

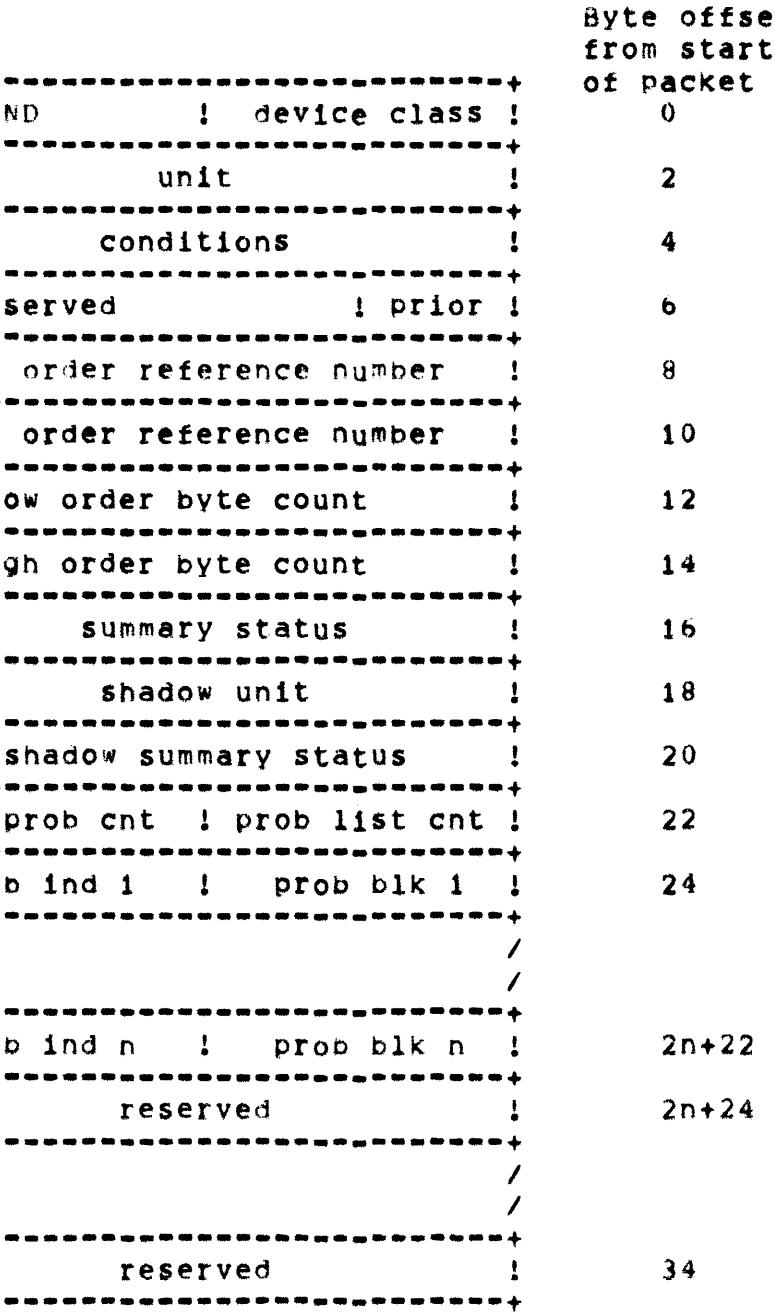
Defined conditions:
---
Error - unit is not a disk

### 4.5.6 Controller-to-Host Messages

This section describes the individual messages the HSC50 controller can send to the host. The numerical values for opcodes, conditions, and other bit specific information (such as status and log blocks) are in the tables at the end of the section.

#### 4.5.6.1 END -

Format:



Allowable units:
---

Same as in the original command packet.

Arguments:
---

Byte Count
Total Problem Block Count
Problem Block List Count
Problem Block List
Summary Status
Shadow Unit Number
Shadow Unit Summary Status

Defined conditions:
---

All condition codes defined in the table in Section 4.5.7.3.1 can be returned.

Function and results:
---

End is a synchronous message to the host signifying that the controller has completed processing the command with the specified reference number. The condition field provides status relative to the command. The summary status fields provide status relative to tne units. See Section 4.5.7.3 for a specification of the conditions field. See Section 4.5.7.4 for a specification of the summmary status fields.

Special Comments:
---

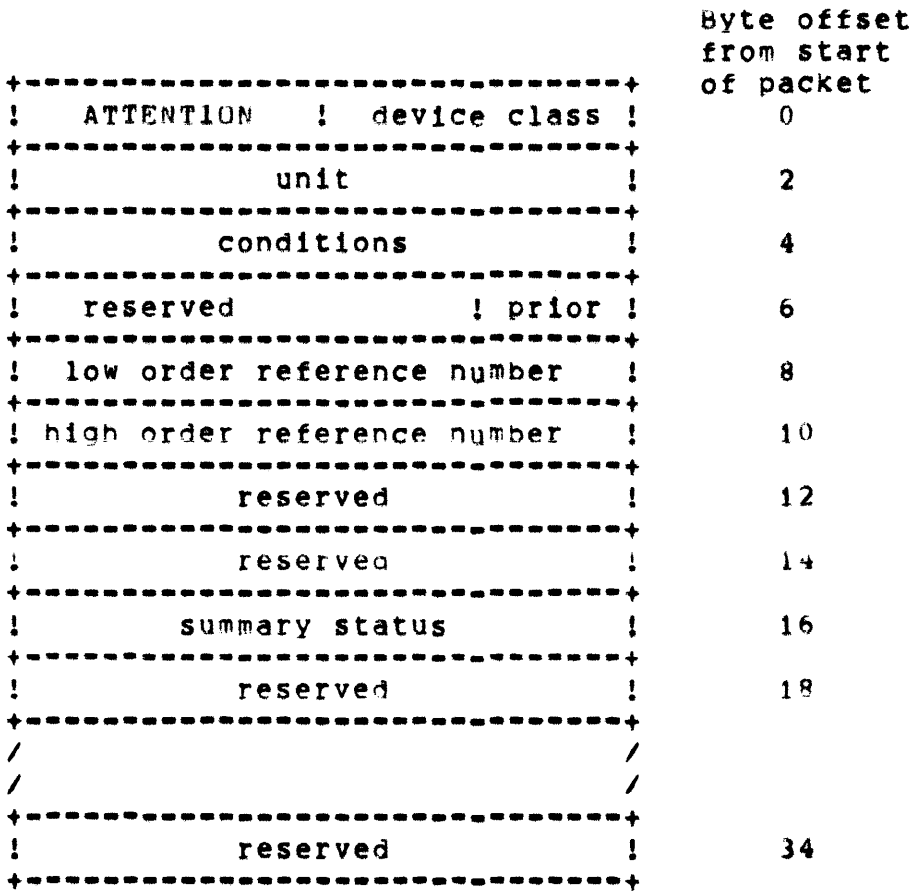
The host must be prepared to receive END packets for which there is no outstanding command. This would be an error on the part of the controller, but the class driver should not "crash". Recommended procedure is to consider receipt of an unexpected END packet as equivalent to command timeout.

The problem block list is the set of blocks in error or which caused tne ECC tnreshold to be exceeded during reading. Tne total problem block count is the number of blocks that had problems. The problem list count is the number of blocks which had problems and are identified in the END packet. The problem indicator has bit 7 set if tnere was a hard error and bit 6 set if the block was on the shadow unit. Bits 0-5 are reserved. The problem block byte is the relative block number from the

starting LBN .

4.5.6.2 ATTENTION -

Format:




Allowable Units:

Any

Arguments:

Summary Status

Defined Conditions:

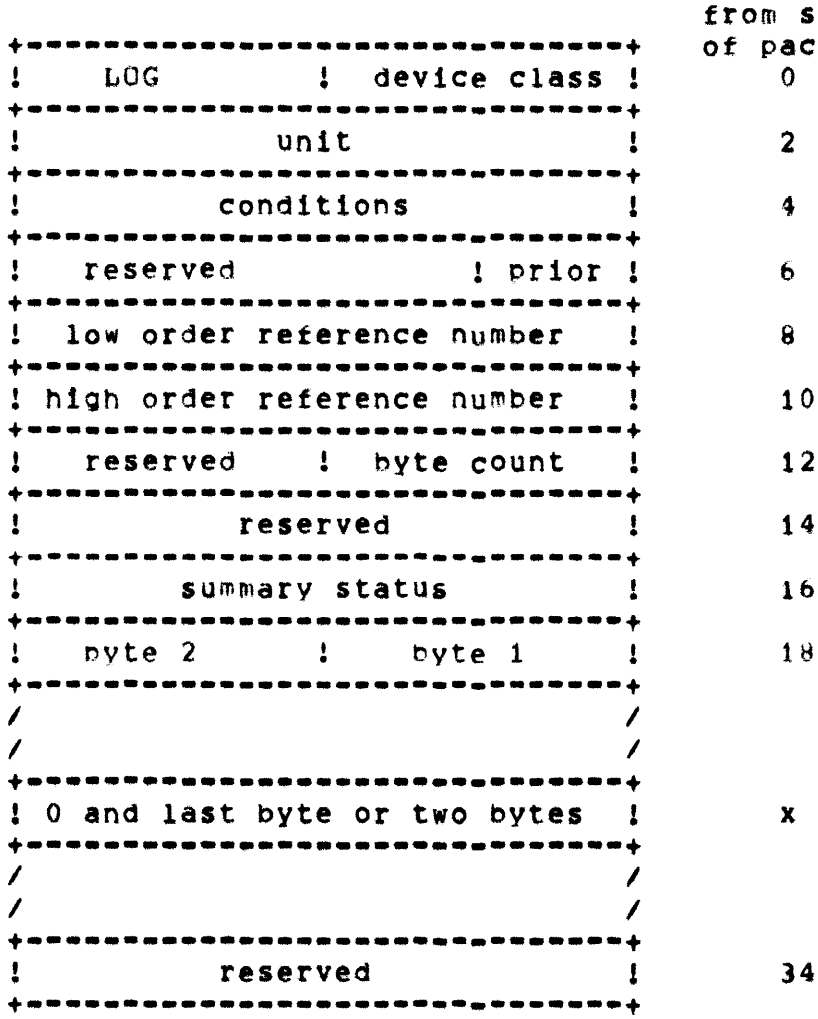
Any of the conditions defined in Section 4.5.7.3.2.

Function and Results:

ATTENTION is an asynchronous message from the controller to the host and is not acknowledged by the host with any corresponding message. ATTENTION signals the host of a change in the specified unit' status or characteristics. The conditions and summary status contain the information specifying the change and current status.

4.5.6.3 LOG - -

Format:




Allowable Units:

Any

Arguments:

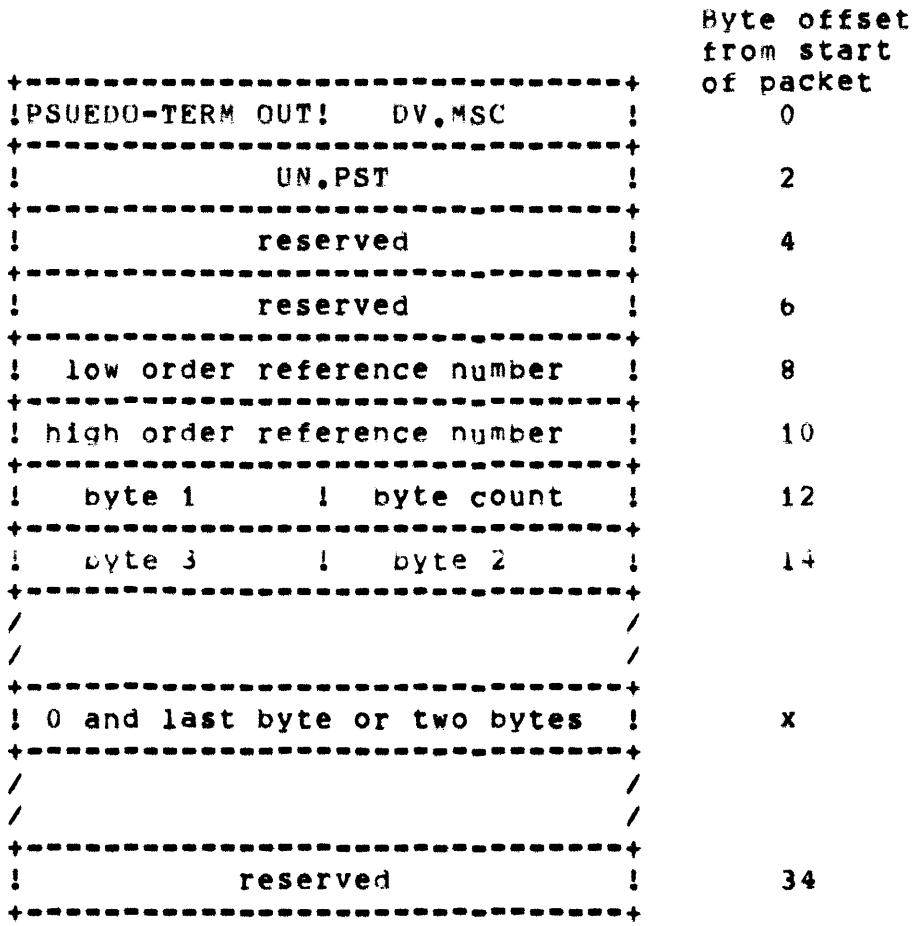
Summary Status
Log Information

Function and Results:

LOG is an asynchronous message from the controller and is not acknowledged with a corresponding message. The error logging information may be the result of processing a specified command (identified by the reference number) or may be unrelated to any specific command (reference number is zero).

### 4.5.6.4 PSUEDO-TERM OUT - -

Format:




Allowable Unit:

Psuedo-terminal unit number

Arguments:

Byte Count
ASCII String

Function and Results:

PSUEDO-TERM OUT provides the host with a string to be displayed on the "psuedo-terminal".

Special Comments:
---

These messages are normally not sent. A host must set a controller characteristic to enable them.

## 4.5.7 Block and Symbol Definition Tables

### 4.5.7.1 Opcodes - -

| Opcode | Hex |
|---|---|
| ABORT | - C3 |
| ATTENTION | - F3 |
| COMPARE | - 33 |
| DOWLINE LOAD | - A3 |
| END | - F5 |
| EXECUTE | - A5 |
| FLUSH | - 63 |
| GET CHAR | - 93 |
| GET STATUS | - 95 |
| INIT | - C5 |
| INVALIDATE | - 65 |
| LOG | - F6 |
| NOP | - 35 |
| POSITION | - 53 |
| PSUEDO-TERM IN | - A6 |
| PSUEDO-TERM OUT | - F9 |
| READ | - 36 |
| SET CHAR | - 96 |
| SET STATUS | - 99 |
| WRITE | - 39 |
| WRITE TPMRKS | - 55 |

### 4.5.7.2 Modifiers -

| Mnemonic | Bit |
|---|---|
| PO.FIL | 15 |
| PO.BOT | 14 |
| PO.UNL | 13 |
| PO.LET | 12 |
| RD.NEC | 15 |
| RD.REV | 14 |
| RD.CMP | 13 |
| RD.NOC | 12 |
| WR.CMP | 13 |
| WR.BCK | 11 |
| WR.NOC | 12 |
| WR.NOS | 10 |
| WR.REP | 9 |

```
WT.MRK     11
WT.EGP     10
WT.DSE      9
WT.LET     12

MD.EXT      0 (modifiers escape)
```

4.5.7.3 Condition Codes -

4.5.7.3.1 Condition Codes for END -

```
Bit 15 - Succeed(0)/Failure(1)
Bit 14 - 1 if a problem block list included in END packet
Bit 13 - 1 if ECC threshold exceeded
Bit 12 - 1 if other recording error (header compare or EDC)
Bit 11 - 1 if unit context incompatible with command
Bit 10 - 1 if command packet invalid
Bit  9 - 1 if command failure in execution(COMPARE)
Bit  8 - Reserved
Bits 7-0 - Error code number specific to one of the above errors
```

4.5.7.3.2 Condition Codes for ATTENTION -

```
Bits 15-8 - Reserved
Bits  7-0 - Attention code
                 REINIT - 3
                 CH-CHG - 5 (characteristic change)
                 ST-CHG - 6 (status change)
                 SH-CPY - 9 (shadow copy complete,
                             units in bytes 12-13)
```

4.5.7.4 Summary Status Format -

```
Bits 15-14 - General status
        00 - Unit online
        01 - Unit available
        10 - Unit offline
        11 - Unit non-existent
Bit 13 - Unit write protected
```

Bit 12 - Unit at BOT (tape class only)
Bit 11 - Unit at EOT (tape class only)
Bit 10 - Other unit status problem (GET STATUS for detail)
Bits 9-0 - Reserved

4.5.7.5 Log Information Format -
The log information is a counted ASCII string.

4.5.7.6 Status Format -
Bytes 12-13 - Same as summary status format
Byte 14 -
Bit 0 - write protected by program
Bit 1 - write protected physically
Bit 2 - pack spin down / tape unloading
Byte 15 - Current tape density
Same as in tape characteristics block

4.5.7.7 Characteristics Format -

4.5.7.7.1 Controller Characteristics Format -

For each host:
Byte 12 - Error and log control
Bit 0 - Send logs for this host
Bit 1 - Send logs for all hosts
Bit 2 - Configuration change reporting enabled
Bit 3 - Psuedo-terminal enabled
Byte 13 - Thesholds
Bits 0-3 - ECC threshold
A value of 0 indicates to use the serious level threshold of each unit.
Bits 4-7 - Log threshold
Byte 14 - Reserved

Common to all hosts
Configuration
Byte 15 - Number of hosts
Byte 16 - Number of disks
Byte 17 - Number of tapes
Byte 18-19 - Number of buffers

Byte 20-21 - Number of cache blocks
Shadow pairs
Byte 22 - Shadow copy control
Bit 0 - Copy for first pair
Bit 2 - Copy for second pair
Bit 4 - Copy for third pair
Bit 6 - Copy for fourth pair
Byte 23 - Reserved
Bytes 24-25 - First pair of units \
Bytes 26-27 - Second pair of units ! low order 8 bits
Bytes 28-29 - Third pair of units ! of each unit
Bytes 30-31 - Fourth pair of units /

Bytes 32-33 - Reserved

Bytes 34-35 - Diagnostic control

4.5.7.7.2 Disk Characteristics Format -

Geometry parameters
Bytes 12-15 - maximum LBN
Bytes 16-19 - revector table LBN
Bytes 20-21 - tracks/cylinder
Bytes 22-23 - LBN/track(s-r of SDI specification)
Bytes 24-25 - RBN/track(r of SDI specification)
Byte 26
Bit 0 - Dual ported unit
Bit 1 - 512/576 indicator
Bit 2 - Removable pack
Bit 3 - Reserved
Bits 4-7 - Serious ECC threshold

Byte 27 - Performance statistic

Byte 28 - Unit plug number

Bytes 29-30 - Unit serial number

Byte 31 - Flags
Bit 0 - Compare on all reads
Bit 1 - Compare on all writes
Bits 2-7 - Reserved

Bytes 32-35 Diagnostic control

4.5.7.7.3 Tape Characteristics Format -

Bytes 12-23 - Reserved

Byte 24 - Possible density settings
- Bit 0 - 556 bpi
- Bit 1 - 800 bpi
- Bit 2 - 1600 bpi
- Bit 3 - 6250 bpi
- Bits 4-7 - Reserved

Byte 25 - Reserved

Byte 26
- Bit 0 - Dual ported unit
- Bits 1-7 - Reserved

Byte 27 - Performance statistic

Byte 28 - Unit plug number

Bytes 29-30 - Unit serial number

Byte 31 - Flags
- Bit 0 - Compare on all reads
- Bit 1 - Compare on all writes
- Bits 2-7 - Reserved

Bytes 32-35 Diagnostic control

4.5.7.8 Device and Unit Numbers -

DV.MSC = 00 (controller)
DV.DSK = 01 (disk class device)
DV.TAP = 02 (tape class device)

UN.CTR = 00FF (controller)
UN.PST = 00FE (psuedo-terminal, DV.CTR)

4.5.7.9 Priority -

PR.NRM = 0
PR.HI = 1