MSCP Basic Disk Functions Manual
AA-L619A-TK Version 1.2

A part of UDA50 Programmer's
Doc. Kit
QP-905-GZ

First Edition

April, 1982

CHAPTER 1

INTRODUCTION

1.1 Overview of MSCP Subsystem

Mass Storage Control Protocol (MSCP) is the protocol used by a family of mass storage controllers and devices designed and built by Digital Equipment Corporation. In a system that uses an MSCP storage subsystem, the controller contains intelligence to perform the detailed I/O handling tasks. This arrangement allows the host to simply send command messages (requests for reads or writes) to the controller and receive response messages back from the controller. The host does not concern itself with details such as device type, media geometry, media format, error recovery, etc.

The host uses two levels of software to accomplish its tasks. The higher level is called a "class driver". The class driver's knowledge of devices is limited to the device class (such as disks) and their capacity. The class driver does not have to know the detailed nature of the communications link (I/O bus), controller, or devices that are being used.

The second level of host software is called a "port driver". The port driver passes messages to/from the communications link or bus. It is not aware of the messages' meaning. The port driver does have to know the exact nature of the communications link or bus (communications mechanism).

In the controller architecture, there are also two levels of software. The lower of these two is also a "port driver" and, like the port driver in the host, is concerned only with passing messages on and off of the bus. The higher level of controller software is the "MSCP Server". It constitutes the intelligence of the controller and therefore defines the functionality of the controller.

The MSCP server concerns itself with determining the number of devices, their type, geometry, unit number, availability, status, etc. The MSCP server receives requests from the host and sends responses to the host. It optimizes the requests, performs the operations, transfers the data to/from the host, transfers the data to/from the device, and buffers the data as necessary. The

MSCP server performs error detection and recovery, and reports any significant errors to the host.

Because the MSCP server handles the error detection and recovery by itself, the host sees a "perfect media", an important characteristic of an MSCP subsystem. That is, the host need only report errors to higher level (user) software, as the MSCP server performs all error recovery and media defect (bad block) handling.

The host's class driver and the controller's MSCP server route their messages through the path of the two port drivers and a hardware interconnect. This is their physical connection. However, their logical communication is a direct connection because the port driver details are below their level of concern. Therefore, there are two paths to consider, a physical message path and a logical MSCP connection. This is illustrated in Figure 1.

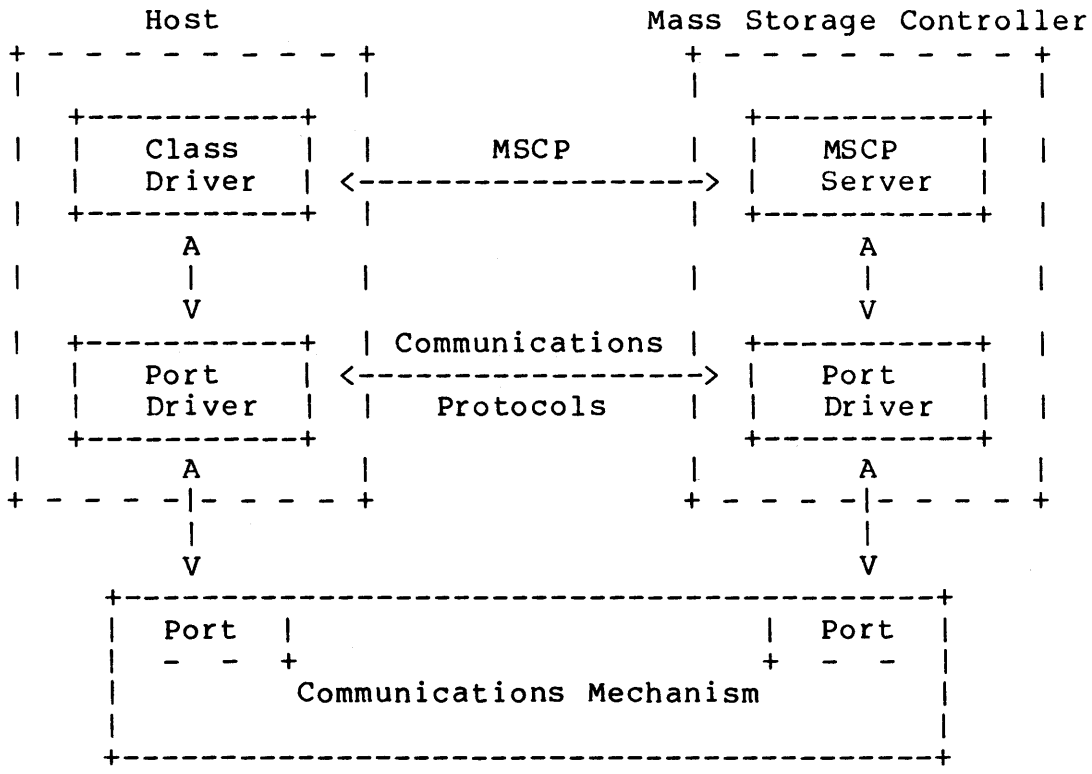


Figure 1 Example System

In summary, an MSCP subsystem is characterized by an intelligent controller that provides the host with the view of perfect media. It is further characterized by host independence from a specific bus, controller, or device type.

1.2 Purpose

The purpose of this manual is to provide information on the rules of MSCP to the detail necessary for writing a host class driver.

1.3 Method of Presentation

The method of presentation used in this manual is:

- to define new terms and concepts.
- list the responsibilities of the class driver.
- list the responsibilities of the MSCP server that are applicable to the class driver.
- list the responsibilities of the storage unit that are applicable to the class driver.
- explain each command and response message.
- explain each error message.
- provide appropriate tables of consolidated information.

1.4 Scope

The scope of this manual is limited to the details of the MSCP itself and does not provide information on any specific type of host processor or any specific operating system. It does not assume any particular bus, controller, device type, host, or port driver implementation.

CHAPTER 2

TERMINOLOGY

Command Categories

All MSCP commands fall into one of four command categories: Immediate commands, Non-Sequential commands, Sequential commands, and Special commands. Each command category has certain constraints on when those commands may be executed, thus limiting the scope of controller optimizations. See Section "Command Catagories and Execution Order".

Controller Timeout Interval

A time interval, measured in seconds, supplied by the controller or MSCP server in the SET CONTROLLER CHARACTERISTICS command's end message. Controllers or MSCP servers guarantee that they will complete all Immediate commands plus some measurable amount of useful work on their oldest outstanding non-Immediate command within the controller timeout interval. See Section "Command Timeouts".

Forced Error

A data error (in a disk block) that has been deliberately caused by use of the "Force Error" command modifier. Used to indicate that the data in the block is of questionable validity. For example, an unrecoverable error occurred when the data was copied from some other block.

Forced Error Indicator

The logical flag, present in each disk block, used to record the presence of a Forced Error. Depending upon the detailed, low level format of the disk device, this may be implemented either as an actual bit flag or as a special pattern (such as the complement of the normal value) of error correcting and/or error detecting codes.

Immediate Commands

Commands that MSCP servers should execute immediately, without waiting for any other commands to complete. Immediate commands are typically status inquiries, and must be completed within the controller timeout interval.

Non-Sequential Commands

Commands whose execution order MSCP servers may rearrange, in order to optimize performance. The optimization may not move a non-sequential command past the barrier imposed by a sequential command.

Nugatory

Of little or no consequence: Trifling, Inconsequential. See Section "AVAILABLE Command"

Sequential Commands

Commands that MSCP servers must execute in the exact order that they were received from class drivers. Sequential commands typically change a unit's state or context.

Special Commands

Commands that have both the execution order constraints of non-sequential commands plus certain special, command dependent execution order constraints.

CHAPTER 3

CLASS DRIVER / MSCP SERVER COMMUNICATIONS

3.1 Connection

Host class drivers use the host port driver to communicate with MSCP servers in controllers. MSCP servers similarly use the controller port driver to communicate with class drivers in hosts. This communication takes place across a link called a connection.

The state of the connection is directly equivalent to the state of the controller or MSCP server with respect to the class driver. The controller is "Controller-Online" if and only if the connection is established and functioning. The controller is "Controller-Available" if the connection is not established, but it is believed that it could be established. The controller is "Controller-Offline" if the connections is not established and it is believed that it cannot be established.

Three types of communications services are used across the connection between a class driver and an MSCP server:

- A sequential message communications service, used for MSCP control messages. This service guarantees sequential, duplicate free delivery for all messages sent across the same connection. This service must support messages of at least 48 bytes in length.

- A datagram communication service, used for MSCP error log messages. This service must deliver messages sent on it with very high probability; messages may be delivered out of sequence, lost, or duplicated, but the probability of any of these occurring must be very low. This service must support messages of at least 384 bytes in length.

- A block data communication service, used to move data between hosts and mass storage controllers. This service provides a reliable, efficient method of transferring the contents of a named buffer in one subsystem to a named buffer in another subsystem. Buffers are identified by buffer descriptors, which

identify both the buffer and the subsystem (host) in which the buffer resides.

The communications mechanism or port drivers discard all messages that, at the time a connection is terminated, have been sent or queued to be sent via the sequential message and datagram services but have not yet been delivered. Block data transfers may or may not be aborted when a connection is terminated; if aborted, they may have already been partially completed. Block data transfers from a previous incarnation of a connection are guaranteed to be aborted when the connection is re-established.

Besides using these three communications services directly, MSCP uses the establishment of the connection itself to synchronize class drivers and MSCP servers. Either the class driver or the MSCP server will terminate the connection between them (become "Controller-Available") if it determines that they must re-synchronize with each other. Events that require class driver / MSCP server re-synchronization include certain errors or loss of context by either process. The connection is also terminated, by a port driver, if an unrecoverable communications error occurs. Termination of the connection signals the processes that re-synchronization is necessary; the re-synchronization is accomplished by each process discarding all context regarding outstanding commands or transactions, after which a new connection is established.

Following re-synchronization, commands which were outstanding before the re-synchronization was performed may have completed to an indeterminate extent. Such commands may have never been started, may have been partially completed, or may have been fully completed. The only guarantee is that they are no longer outstanding, implying that the controller is no longer performing work for them and that the class driver will not receive an end message for them. The fact that the controller is no longer performing work for them implies that no state changes or modification of data will take place as a result of such commands.

3.2 Flow Control

Especially critical to MSCP is the concept of flow control and the flow control based requirements that MSCP imposes on class drivers and MSCP servers. These items are discussed below.

Flow control arises from the need to avoid the congestion and/or deadlock which can occur if one process sends messages too quickly to another process. The receiving process must have buffers in which to place the incoming messages; when all such buffers are full, additional messages cannot be handled.

3.2 Flow Control

The datagram communications service does not use flow control. If no buffers are available, incoming datagrams will be discarded. Thus the characteristic that the datagram service does not guarantee delivery, instead only assuring a high probability of delivery. This high probability is dependent upon the receiver (i.e., the class driver) always having buffers queued for incoming datagrams.

The sequential message communications service does use flow control. When a potential receiving process queues a buffer for receiving messages on a connection, the presence of this buffer is communicated (via the underlying communications service) to the potential sending process at the other end of the connection. This message notifying the potential sending process of the queued buffer grants the sending process a credit, which is the priviledge to send a message. Therefore messages will only be sent when the sending process knows that the receiving process has queued a buffer into which the message can be received, ensuring that the receiving process will be able to handle the message.

A typical implementation of flow control will be somewhat as follows. The port driver maintains a counter on behalf of each process participating in a connection. That counter holds the process's current credit balance -- i.e., the number of receive buffers that its partner has queued less the number of messages that it has sent. Every time the process's partner queues a receive buffer, a message is sent causing the counter to be incremented. Every time the process sends a message, the counter is decremented. Messages may only be sent when the counter (credit balance) is greater than zero, thus guaranteeing that the counter will never be negative. Indeed, we will see later that some messages require that the counter be greater than a threshhold larger than zero.

Due to the inherent asynchrony of communications between multiple processes or subsystems, revoking or canceling a previously queued receive buffer is not straightforward. The problem is that the buffer cannot be revoked and returned to the receiving process until after the sending process has acknowledged its revocation, as otherwise the sending process may attempt to send a message that requires the revoked buffer. Therefore the algorithm for revoking receive buffers is as follows:

1. The revoking process (the process which originally queued the receive buffers) requests that some number of buffers be revoked.

2. The revocation request is communicated to the revoking process's partner (actually to its port driver).

3. The revoking process's partner (actually its port driver) compares the number of buffers to be revoked against its current credit balance and a threshhold. If the requested number of buffers / credits can be revoked (i.e., subtracted from the credit balance) without lowering the credit balance below the threshhold, then all of them will be revoked. Otherwise, if the credit balance is above the threshhold, it is set to the threshhold and the difference between its former value and the threshhold is the number of buffers / credits actually revoked. If the credit balance is already at or below the threshhold, then it stays the same and no buffers / credits are revoked.

4. The actual number of buffers / credits revoked is communicated back to the revoking process's port driver.

5. The revoking process's port driver returns the buffers actually revoked to the revoking process.

If a threshhold of zero is used, the revoking or receiving process can always get back all of its buffers. The fact that an attempted revocation failed implies that the buffers have already been returned to the process, since messages have been received into them.

The above algorithm uses a threshhold to prevent revocation below some lower limit. The mechanism by which this threshhold is obtained is not critical to either MSCP or to the above algorithm. The rules below are phrased as if the threshhold were supplied by the revoking process as part of the revocation request. This is purely to simplify the wording of those rules; often the threshholds will be constants determined when a connection is established. In such an implementation a threshhold of zero should be used when the class driver is revoking credits it has granted to the MSCP server and a threshhold of one should be used when the MSCP server is revoking credits it has granted to the class driver.

MSCP is only concerned with credits required by the sequenced message service. Some communications services may require credits for the block data communication service as well. Any such credits are invisible to MSCP, being communications service dependent, and must be provided in addition to the credits required by the rules below.

Note that the above discussion merely describes a conceptual model for flow control within the sequential message communications service. There is no requirement that flow control actually be implemented this way, provided that the results are the same. For example, almost all implementations will carry credit information in a header added to messages and processed by the receiving process's port driver, rather than communicating credits with separate messages. Some extremely

well behaved communications mechanisms may not need to implement explicit flow control at all, since the underlying communications mechanism may provide it implicitly.

3.3 Class Driver Responsibilities

Given the above model for flow control, we can state the requirements MSCP places on class drivers and MSCP servers. Class drivers must obey the following rules:

1. All MSCP commands fall into one of several categories; for this discussion we distinguish between Immediate commands (one specific command category) and non-Immediate commands (the union of all other command categories). When the class driver's credit balance is zero, the class driver may not issue any commands. When it is one, the class driver may only issue Immediate commands. When it is two or larger, the class driver may issue both Immediate and non-Immediate commands. If the class driver's credit balance is one and there is a GET COMMAND STATUS command waiting to be issued for the command timeout algorithm, then the class driver must issue that GET COMMAND STATUS command as the next command.

   In essence, this rule means that the class driver must reserve one credit for the exclusive use of the command timeout algorithm. This credit may be "borrowed" for issueing Immediate commands, since such command always complete quickly. The goal is to guarantee that the command timeout algorithm will always be able to promptly issue a GET COMMAND STATUS command.

2. The class driver must queue a receive buffer for each command that it sends to an MSCP server. The receive buffer will be used to hold the command's end message. The receive buffer must be queued either before the command is sent or as part of an atomic (indivisible) action that includes sending the command. The important point is that the MSCP server must receive the credit for the receive buffer either before or concurrently with receiving the command.

3. In addition to queueing receive buffers for end messages, class drivers that enable attention messages must queue at least one receive buffer in which to receive attention messages. Such a receive buffer must be queued before the class driver enables attention messages -- i.e., before the class driver sends a SET CONTROLLER CHARACTERISTICS command that enables attention messages. Additional receive buffers may be queued at any time.

4. Upon receiving an attention message, the class driver must immediately queue another (or the same) receive buffer back for more attention messages. The only resource that the class driver may require between receiving an attention message and queueing another receive buffer is host CPU cycles; the class driver may not require that it be able to send or receive any other messages or wait for any I/O to complete before queueing another receive buffer. This effectively requires that all code and data structures needed to process attention messages must be permanently resident in physical memory, so that an incoming attention message can be immediately processed and the buffer in which it was received immediately re-queued.

5. With one exception, the class driver must never revoke receive buffers. The one exception is after disabling attention messages -- i.e., after receiving the end message for the SET CONTROLLER CHARACTERISTICS command that disabled attention messages. At that time the class driver may revoke as many buffers as it has queued for attention messages (i.e., total number of buffers queued less number of outstanding commands). This revocation should specify a threshhold of zero. Note that this revocation is guaranteed to succeed if the MSCP server is operating correctly.

Failure of a class driver to follow the above rules may lead to controller deadlock, command timeouts, or the controller not obeying its rules given below. Any connection or process in the controller's subsystem may be affected, rather than just the MSCP server with which the class driver is communicating. Note that class drivers that enable error logging must also keep datagram buffers queued so that they can receive error log messages. Not keeping datagram buffers queued may result in loss of error log messages.

3.4 MSCP Server Responsibilities

The rules for MSCP servers vary depending upon certain characteristics of the server. There is one general set of rules, plus a simplification that certain classes of servers may follow. In all cases, an MSCP server's implementation of these rules may require, for its correct operation (and thus adherence to these rules), that class drivers correctly follow the rules given for them above. The general set of rules, which must be followed by all MSCP servers that aren't in any of the special cases identified later, are as follows:

1. So long as it is "Controller-Online" to a class driver, an MSCP server must ensure that the sum of the number of commands that are outstanding from that class driver plus the number of unused receive buffers / credits that it has granted to that class driver is never lower than the values given below. That is, the sum of the actual and potential outstanding commands must be at least the values below. Between the time that the controller becomes "Controller-Online" and the completion of the first SET CONTROLLER CHARACTERISTICS command, this sum must be at least one. Following the completion of the first SET CONTROLLER CHARACTERISTICS command, and so long as the controller remains "Controller-Online", this sum must be at least two. The first unit of this sum allows the class driver to issue Immediate commands; any excess, beyond the value one, can be used to issue "real" commands such as data transfers. Note that these requirements imply that, following a SET CONTROLLER CHARACTERISTICS command, the MSCP server must either grant a minimum of two receive buffers / credits to the class driver or else terminate the connection (become "Controller-Available") with the class driver.

2. So long as it is "Controller-Online" to a class driver, an MSCP server must ensure that the sum of the number of immediate commands that are outstanding from that class driver plus the number of unused receive buffers / credits that it has granted to that class driver is always at least one. That is, the sum of the actual and potential outstanding Immediate commands must be at least one. This is in addition to requirement 1 above.

3. An MSCP server may revoke receive buffers or credits at any time so long as it continues to meet requirements 1 and 2 above. Note that, in order to meet requirement 2, the sum of the threshhold used for the revoke request plus the number of outstanding Immediate commands (from that class driver) must be at least one. This restriction will typically be met by always using a threshhold of one.

4. An MSCP server must keep track of the excess, if any, of its credit balance over the number of outstanding commands. The server may only issue attention messages when this excess is greater than zero. Attention messages that cannot be issued immediately should be saved until credits are available with which to issue them. The controller must continue to accept new commands, process outstanding commands, and issue end messages for completed commands while attention messages are being saved. If the conditions that triggered the generation of an attention message disappear before that attention message can be issued (sent), then the attention message may or may not still have to be sent;

see the individual attention message descriptions.

End messages for all types of commands should be issued as soon as the command completes. Note that the rules for class drivers ensure that the MSCP server always has sufficient credits with which to issue end messages, so the MSCP server need not check its credit balance before issuing end messages.

MSCP servers that limit the rate at which they generate attention messages can replace rule 4 above with a simpler rule. This alternate rule may be used, at the server's option, by any MSCP server that will generate no more than an average of two attention messages per second, averaged over the controller timeout interval (see Section "Command Timeouts"). That is, if the controller timeout interval is N seconds, then the server will generate no more than N*2 attention messages within any N second period. The MSCP server may assume, in determining its maximum attention message rate, that human operators do not engage in pathological activity. That is, it may assume that cases such as an operator continuously actuating the Run/Stop switches on one or more drives will never occur. Any MSCP server that can meet this attention message rate restriction can substitute the following alternate rule for rule 4 above:

4'. An MSCP server may issue attention messages and end messages whenever its credit balance is greater than zero. Attention messages and end messages that cannot be issued immediately should be saved until credits are available with which to issue them. If the conditions that triggered the generation of an attention message disappear before the attention message can be issued (sent), then that attention message may or may not still have to be sent; see the individual attention message descriptions. Saved end messages must always be sent, unless the MSCP server first becomes "Controller-Available" (i.e., terminates its connections with the class driver), in which case the saved end messages must not be sent. Note that end messages must always be sent in the order that their corresponding commands completed.

An MSCP server may deadlock or cease operating on a connection whenever it has saved attention messages or end messages waiting to be issued on that connection. The only operations that it must perform when it has such messages waiting is to accept incoming credit notifications, send the waiting messages when credits become available, and resume normal operation when all waiting messages have been sent. Note that accepting incoming credit notifications will often require that the MSCP server also accept new commands, although it need not begin processing of those new commands until

all waiting messages have been sent. Note also that only the one MSCP server may deadlock or cease operating; other processes (i.e., other MSCP servers) in the same subsystem or controller and other connections to the same MSCP server must not be affected.

CHAPTER 4

ALGORITHMS AND USAGE RULES

4.1 Controller States

The controller may be in any of three states relative to a host class driver. The controller may be in a different state relative to each host or each class driver. The controller states are:

Controller-Offline

> A controller is "Controller-Offline" to a class driver whenever it is not available to that class driver and cannot perform any operations on its behalf. Possible causes include inoperative hardware or an operator disabling the controller. A controller is "Controller-Offline" exactly when it is not possible to establish a connection between the class driver and the MSCP server within the controller. Note that a controller may be "Controller-Offline" to some of a host's class drivers yet be "Controller-Available" or "Controller-Online" to others.

Controller-Available

> A controller is "Controller-Available" to a class driver whenever it could perform operations for that class driver but the driver has not yet synchronized with the controller. A controller is "Controller-Available" exactly when it would be possible to establish a connection between the class driver and the MSCP server within the controller, but no connection has yet been established.

Controller-Online

> A controller is "Controller-Online" to a class driver whenever it can both perform operations for that class driver and the driver has synchronized with the controller. A controller is "Controller-OnlineOR" exactly when a connection exists between the class driver and the MSCP server within the controller. This is the state used for normal operation.

Strictly speaking, the term "controller state" is a misnomer. The states described above actually exist between an individual class driver and an individual MSCP server. A host may have several class drivers and a controller subsystem may have several MSCP servers. Note also that the controller state (MSCP server state?) is distinct from the state of any units connected to the controller.

An MSCP server (controller) enters the "Controller-Offline" state relative to a host whenever the MSCP server ceases to function or otherwise becomes unable to perform operations for the host. Possible causes include:

1. Controller hardware, software, or power failure.

2. Controller initialization, either requested or spontaneous.

3. An operator (typically Field Service) disables all or part of the controller.

4. Communications mechanism failures.

5. The controller has not been built yet, the controller is still in its shipping crate, or it has otherwise not yet been installed.

An MSCP server enters the "Controller-Available" state relative to a host class driver when:

1. The controller or MSCP server is "Controller-Offline", and all causes of it being "Controller-Offline" are removed.

2. The MSCP server is "Controller-Online", and the MSCP server cannot successfully send a control message (i.e., an MSCP end or attention message) to the host class driver.

3. The MSCP server is "Controller-Online", and the host access timeout expires (see Section "Host Access Timeouts").

4. The MSCP server is "Controller-Online", and the MSCP server receives an invalid command from the host. Note that this transition to "Controller-Available" is optional, and therefore controller dependent.

5. The host class driver terminates the connection between the class driver and the MSCP server.

6. A port driver or the communications mechanism terminates the connection between the class driver and the MSCP server, generally due to a communications error.

The port driver should inform the class driver whenever the MSCP server enters the "Controller-Available" state. How the port driver obtains this information is communications mechanism dependent. Note that the notification that the controller has become "Controller-Available" is not necessarily prompt. In particular, with some communications mechanisms the notification may not occur until the next time the class driver issues a command to the controller. Furthermore, the port driver need not notify the class driver at all if a compound (multiple) error is associated with the MSCP server becoming "Controller-Available". In such a case the class driver will ultimately become aware of the state change when its Command Timeout expires.

Since no connection exists to an MSCP server that is "Controller-Offline" or "Controller-Available", the communications mechanism will either reject or discard any messages (commands) that a class driver attempts to send to it. An MSCP server that becomes "Controller-Offline" or "Controller-Available" may either abort commands in progress or else continue processing the commands that it has already received. However, if a Sequential command from a given connection is aborted, then all subsequently received non-Immediate commands from the same connection must also be aborted (i.e., must never begin processing).

Typically, the MSCP server will continue processing outstanding commands until it "notices" that the connection to the class driver has been terminated, at which point it will abort any commands still outstanding. Note that the MSCP server must guarantee that all outstanding commands have either been completed or aborted -- i.e., that there are no outstanding commands -- before it completes a transition from "Controller-Available" to "Controller-Online".

The MSCP server enters the "Controller-Online" state relative to a host class driver upon successful synchronization with the class driver. The class driver synchronizes with the MSCP server by establishing a connection with the MSCP server. Note that the MSCP server must guarantee that there are no outstanding commands "leftover" from a previous incarnation of the connection before it allows the new incarnation of the connection to be established and enters the "Controller-Online" state.

4.2 Controls and Indicators

All storage units used with MSCP must have the following controls and indicators:

- Unit number select mechanism.
- Unit number display mechanism.
- Run/Stop switch.
- Write Protect switch or mechanism.
- Write Protect Status indicator.

The unit number select mechanism on an MSCP storage unit must be capable of specifying any unit number in the range Ø through 251 inclusive. The unit number select mechanism must operate without host intervention and must preserve unit numbers across power failures and other losses of context.

Alteration of the unit number must be possible in the field. That is, the unit number must be alterable by Field Service, both when a device is first installed and subsequently when a system is reconfigured. The preferred unit select mechanism is a removable unit number plug, allowing the unit number to be altered by users as well as by Field Service. Alternatives to unit number plugs, however, are acceptable so long as they can be altered by Field Service and they provide the full Ø through 251 unit number range.

A single unit number select mechanism may be shared by the units of a multi-unit drive. Units that share a unit number select mechanism always have consecutive unit numbers. If exactly two units share a unit number select mechanism, it is acceptable for the first unit to always have an even unit number and the last unit to always have an odd unit number. If exactly N units share a unit number select mechanism, it is acceptable for the first unit's unit number to always be a multiple of N, the next unit's unit number to always be a multiple of N plus 1, etc.

The unit number display mechanism on an MSCP storage unit must display the unit number(s) specified by the unit number select mechanism. The display must be visible to normal (non-field service) human operators. If the unit number select mechanism is a removable unit plug, then the unit number display mechanism is merely a number printed on the plug. If several units share a unit number select mechanism, then they may also share a unit number display mechanism.

The Run/Stop switch must be alterable by normal (non-field service) human operators to allow or disallow host access to the unit(s). When in the Run position, this switch indicates that hosts should be allowed to access the unit(s). When in the Stop position, this switch indicates that hosts should not be allowed to access the unit(s), and that, if the unit(s) have removable media, human operators should be allowed to remove the units' media (i.e., that the unit should be spun-down). A single Run/Stop switch may be shared by any units of a multi-unit drive that share a spindle (i.e., that must be spun-up and spun-down together).

The write protect switch or mechanism must either be an operator accessible switch or else some kind of mechanical deformation of the media, such as a tape write-ring or the write-lockout tab on a cassette. In either case, it must be alterable by normal (non-field service) human operators. A separate write protect switch or mechanism must be provided for each unit of a multi-unit drive. When actuated, the write protect switch or mechanism prevents write access to the unit by hosts and controllers. In the case of a write protect switch, the transition to the write protected state must be "smooth" rather than immediate; see Section "Write Protection".

The write protect status display mechanism must display the write protect status of the unit. A separate write protect status display mechanism must be provided for each unit of a multi-unit drive. The write protect status display must be user visible while a volume is mounted in the unit. That is, the user must not be required to remove the volume from the unit to determine whether or not it is write protected.

The preferred write protect status display mechanism is a light, located within the write protect switch, that is on whenever the unit is write protected. The light must be lit regardless of the reason for the unit being write protected -- i.e., it must be lit when the unit is Software Write Protected (see Section "Write Protection"), as well as when the unit is Hardware Write Protected due to its write protect mechanism being activated.

4.3 Unit States

Each unit may be in one of three states relative to each class driver that is "Controller-Online" to an MSCP server. (Actually, it is really each unit number that these states apply to, rather than to a unit proper). Each unit may be in a different state relative to each "Controller-Online" class driver. The unit states are:

Unit-Offline

A unit is "Unit-Offline" whenever it is unable to satisfy

normal host requests. Except for status queries, MSCP commands addressed to a unit that is "Unit-Offline" will be rejected. Furthermore, some device characteristics may not be available to status queries.

Unit-Available

A unit is "Unit-Available" whenever it would be able to satisfy normal host requests, except that the host has not yet issued an ONLINE command to bring the unit "Unit-Online".

Unit-Online

A unit is "Unit-Online" whenever it is able to satisfy normal host requests and the host has issued a successfull ONLINE command.

A unit's state is meaningless with respect to a class driver that is not "Controller-Online" to the MSCP server or when no class driver is "Controller-Online" to the MSCP server.

The "Unit-Offline" state has six sub-states, related to the exact reason for the unit being "Unit-Offline". These substates are:

1. Unit inoperative. Some fatal error condition in the drive prevents the unit from becoming "Unit-Available" or "Unit-Online".

2. Unit disabled. Field Service or a diagnostic has decided that continued operation of the unit's drive will lead to progressive deterioration and eventual destruction of some portion of the drive or media. The unit has been disabled to prevent its use, and consequent destruction, until Field Service can repair the problem. This cause of the unit being "Unit-Offline" can be overridden, and the unit brought "Unit-Online", by use of the "Allow Self Destruction" modifier to the ONLINE command. Note that some devices may have no way of detecting progressive deterioration, and consequently will never enter this sub-state.

3. Unit known. The controller knows that the specified unit (i.e., a unit with the specified unit number) exists, but the unit is "Unit-Offline" for some normal (non-error) condition. Typical causes of this sub-state include the unit's Run/Stop or Load/Unload switch being in the Stop or Unload position or no volume being mounted in the unit.

4. Online to another controller. The specified unit exists and would be "Unit-Available", except that it or some other unit with which it shares an access path (i.e., some other unit on the same multi-unit drive or formatter) is "Unit-Online" to one or more hosts via

another controller. The unit will become "Unit-Available" via this controller if it and all units with which it shares an access path cease being "Unit-Online" to any hosts via that other controller. In terms of the "Unit-Offline" sub-state that is visible to and reported by a controller, a unit that is actually online to another controller will typically oscillate between the "Online to another controller" sub-state and the "Unit unknown" sub-state, where the frequency of the oscillation is determined by the frequency with which DETERMINE ACCESS PATHS commands are issued to the other controller for this unit or any unit with which it shares an access path. See Section "Multi-Access Drives".

5. Duplicate unit numbers. That is, two or more distinct units have the same unit number assigned to them. Controllers must check for duplicate unit numbers across all units of the same device class that are "Unit-Online", that are "Unit-Available", that are "Unit-Offline" solely due to being disabled or known or having a duplicate unit number, or that the controller knows to be online to another controller. A controller may or may not, at its option, include inoperative units when checking for duplicate unit numbers. Controllers must not check units that are unknown (as described below) nor may they check for duplicate unit numbers across different device classes. Note that a duplicate unit number does not affect a unit that is already "Unit-Online".

6. Unit unknown. As far as the controller can determine, no unit exists with the specified unit number.

It is possible for a unit to be "Unit-Offline" for several reasons at the same time (unless the unit is unknown). If a unit has a duplicate unit number and is not inoperative, then the controller must report the duplicate unit number; other causes of the unit being "Unit-Offline" may also be reported at the controller's option. In all other cases the controller must report at least one cause of the unit being "Unit-Offline", but which one it reports and whether or not it reports more than one is optional with the controller.

The fact that a unit is inoperative may not be detectable until a host attempts to bring the unit "Unit-Online". Such units will be treated as and appear to be "Unit-Available" until a host issues an ONLINE command. The ONLINE command will fail, typically with a "Drive Error" status code. At this time either the fact that the unit is inoperative must be recorded in the unit itself or else AVAILABLE attention messages must be suppressed for the unit, exactly as if an AVAILABLE command with the "Spin-down" modifier set had been issued for the unit. In

either case, AVAILABLE attention messages must not be generated for the unit by any controller until a human interacts with the unit or some other event occurs (such as a power failure that may clear the error).

Controllers and/or drives should keep all units that are "Unit-Offline" due to being inoperative, disabled, or having duplicate unit numbers spun-down, except when such units are under the control of a diagnostic. The handling of units that are in fact inoperative, but that the controller and drive believe to be operative, is described in the preceding paragraph.

For the purpose of automatic configuration -- i.e., for the GET UNIT STATUS command with the "Next Unit" modifier -- controllers must acknowledge the existence of all units that are "Unit-Online" or "Unit-Available", or that are "Unit-Offline" solely due to being disabled or known or having duplicate unit numbers. Unknown units must not be acknowledged. Units that are inoperative or online to another controller may or may not be acknowledged at the controller's option.

Controllers must report duplicate unit numbers with a DUPLICATE UNIT NUMBER attention message, then moniter the affected units for the cessation of the duplicate unit number condition. When all units except one have had their unit number changed or have become unknown, the remaining unit becomes "Unit-Available".

Controllers must report units that they know to be online to other controllers with an ACCESS PATH attention message. Section "Multi-Access Drives", describes the detailed circumstances in which this attention message must be sent.

The "Unit-Offline" sub-states are all reported with a single status code; they are partially distinguished via different sub-codes. In addition, the sub-states are distinguished by the functioning of the "Allow Self Destruction" modifier to the ONLINE command, the "Next Unit" modifier to the GET UNIT STATUS command, what unit characteristics are returned by the GET UNIT STATUS, ONLINE, and SET UNIT CHARACTERISTICS commands, and by the DUPLICATE UNIT NUMBER and ACCESS PATH attention messages.

Possible causes of a unit being "Unit-Offline", and the resulting "Unit-Offline" sub-state, include:

1. The unit is "Unit-Online" via another controller. The unit is either unknown or else known to be online to another controller, depending upon how recently the other controller has processed a DETERMINE ACCESS PATHS command for this unit or a unit with which it shares an access path. See Section "Multi-Access Drives".

2. A power failure that affects the unit but not the controller. The unit is unknown.

3. Hardware failure in the unit or in the connection between the unit and the controller. The unit is either unknown or inoperative. Note that the unit may appear to be "Unit-Available" until an ONLINE command is issued for it, at which time it will be recognized as inoperative.

4. Disconnecting the unit from the controller. The unit is unknown.

5. Disabling the unit number select mechanism (i.e., removing the unit number select plug). The unit is unknown.

6. Duplicate unit numbers. Note that this condition will not affect a unit that is already "Unit-Online". This condition has its own sub-state.

7. Duplicate unit identifiers. The unit is inoperative. Note that the controller need not check for duplicate unit identifiers.

8. Disabling the unit with the Run/Stop or Load/Unload switch. The unit is known.

9. Disabling the unit with port selection switches. The unit is unknown.

10. Removal of the unit from service by operator command, typically for diagnostics, formatting, maintenance or repair. The unit is inoperative.

11. An internal controller diagnostic decides the the unit is sick and removes it from service. The unit is disabled.

12. No volume is present in the drive. The unit is known.

13. The unit has not been built yet, the unit is still in its shipping crate, or it has otherwise not been installed yet. The unit is unknown.

In general, a unit that is "Unit-Offline" to one host class driver via a specific MSCP server is "Unit-Offline" for the same reason (same sub-state) to all host class drivers via that same MSCP server. The only exception is a duplicate unit number condition that arises while a unit is "Unit-Online" to one or more class drivers. The unit remains "Unit-Online" to those class drivers to which it is already "Unit-Online", yet is "Unit-Offline" to all other class drivers.

All normal operator initiated transitions to the "Unit-Offline" state must be smooth, rather than abrupt. Attempts to disable a unit with its Run/Stop or Load/Unload switch and/or attempts to dismount (remove) a volume from a unit are, by definition, "normal operator initiated transitions", and must therefore be smooth. Any other action that a human operator would typically use to disable a drive in a non-emergency situation must also be smooth. Note that this does not preclude the existence of some special mechanism for immediately disabling a unit or drive in an emergency, provided that it is not the normal way of disabling a unit or drive. Note that, as used in this paragraph, a command is aborted or rejected if and only if a "Unit-Offline" status code is returned in its end message.

A "smooth" transition to the "Unit-Offline" state implies that all write operations, including multi-block write operations, must either be completed in their entirety or else never begun. To accomplish a smooth transition the controller must complete all write operations (commands) that have already been initiated. Other outstanding commands, including outstanding write operations that haven't been initiated yet and all outstanding read operations, may either be completed or aborted at the controller's option. However, if a Sequential command from a given connection is aborted, then all subsequently received non-Immediate commands from the same connection must also be aborted. Any new commands issued by a host should be rejected.

Note that establishing write protection must also be a smooth transition.

A unit enters the "Unit-Available" state when:

1. The unit is "Unit-Offline" and all causes of the unit being "Unit-Offline" are removed. The unit becomes "Unit-Available" with respect to all "Controller-Online" class drivers.

2. The unit is "Unit-Online" and a host class driver issues an AVAILABLE command for the unit. The unit becomes "Unit-Available" with respect to that class driver. If the "All Class Drivers" modifier was set, the unit also becomes "Unit-Available" with respect to all other class drivers to which it is "Unit-Online" via that MSCP server.

If a class driver has enabled attention messages, the MSCP server uses AVAILABLE attention messages to notify the class driver that a unit has asynchronously become "Unit-Available". If a class driver sends the MSCP server an AVAILABLE command, then the transition to "Unit-Available" is synchronous to that class driver and an AVAILABLE attention message need not be sent to it, although other class drivers are appropriately notified.

The possible causes of an asynchronous transition to "Unit-Available" are as follows:

1. Any transition from "Unit-Offline" to "Unit-Available". This specifically includes the case in which the unit was online to another controller and ceases to be online to that other controller, even if the unit was "Unit-Online" to the same class driver via that other controller.

2. A transition from "Unit-Online" to "Unit-Available", with respect to this class driver, that is caused by some other class driver issuing an AVAILABLE command with the "All Class Drivers" modifier set.

3. Any spontaneous transition from "Unit-Online" to "Unit-Available". That is, any transition from "Unit-Online" to "Unit-Available" that is not caused by an AVAILABLE command.

The one exception to this applies to a unit that becomes "Unit-Available" due to an AVAILABLE command that has the "Spin-down" modifier set or as a side effect of certain errors which also spin-down the unit. Such commands or errors indicate that all class drivers are disinterested in the volume mounted on the unit, so that no class driver should be notified of the transition until an operator mounts a new volume or otherwise interacts with the unit. Therefore the request to spin-down the unit effectively suppresses AVAILABLE attention messages for that unit until an operator interacts with the unit. Note that the messages must be suppressed for all class drivers, MSCP servers, and controllers that may connect to the unit, regardless of which individual MSCP server and controller actually requested that the unit spin-down. This effectively means that, for multi-access drives, the fact that AVAILABLE attention messages are suppressed must be recorded in the drive itself, rather than in the controller.

AVAILABLE attention messages must be suppressed at least until an operator interacts with the unit's drive, the unit becomes "Unit-Online" to any class driver via any MSCP server, or the unit's drive loses context. It is not acceptable for the unit's drive to "lose context" solely to forget that AVAILABLE attention messages have been suppressed; loss of context must be due to some external reason, such as a power failure. The suppression of AVAILABLE attention messages must be cancelled or forgotten if the unit becomes "Unit-Online" to any class driver via any MSCP server or if an operator actuates the unit's Run/Stop or Load/Unload switch, changes the unit number selected by the Unit Number Select Mechanism, or mounts a different volume in the unit. Note that AVAILABLE attention messages are not suppressed (i.e., their suppression is instantly cancelled or forgotten) if, after a class driver issues an AVAILABLE command with the "Spin-down" modifier set, the unit is still "Unit-Online" with

respect to one or more other class drivers.

MSCP servers may send redundant or unnecessary AVAILABLE attention messages at any time, provided that attention messages have been enabled, the messages have not been suppressed as described above, and the extra messages are infrequent enough to avoid causing any significant overhead in either the communications mechanism or a host. It is specifically allowable for an MSCP server to precede every DUPLICATE UNIT NUMBER attention message with an AVAILABLE attention message for the same unit number.

A unit enters the "Unit-Online" state with respect to a class driver upon the successful completion of an ONLINE command issued by that class driver.

4.4 Unit Numbers

As described in Section "Controls and Indicators", all disk accessible via MSCP must have a user visible and Field Service alterable unit number. The characters displayed as the unit number, interpreted as a decimal number, must match the binary value passed in the "unit number" field of MSCP control messages. Multi-unit drives may use a single unit number select mechanism and display for the range of consecutive unit numbers assigned to their units.

MSCP supports unit numbers in the range Ø through 65535 (inclusive). All controllers and units must support unit numbers in the range Ø through 251 (inclusive); unit numbers between 252 and 65535 may be supported or not at the controller's and/or unit's option. This implies that all units accessible via MSCP must have a unit number select mechanism capable of specifying and a unit number display mechanism capable of displaying any unit number in the range Ø through 251. Controllers must treat unsupported unit numbers as if the specified unit is "Unit-Offline" due to being unknown (see preceding section).

The intent of this broad range of unit numbers is that, within a device class, all units that are accessible to a single host must have unique unit numbers. In pursuit of this goal units with duplicate unit numbers are considered to be "Unit-Offline". However, this must be balanced against another goal, namely that transient actions performed on another unit should not be allowed to affect continued host access to a unit that is already "Unit-Online". Controllers must balance these two goals by following the following algorithms:

1. Controllers detect and respond to duplicate unit numbers across all units that are "Unit-Online", that are "Unit-Available", that are "Unit-Offline" solely due to being disabled or known or having duplicate unit

numbers, or that the controller knows to be online to another controller. Units that are "Unit-Offline" due to being unknown must not be considered for duplicate unit number detection. Controllers may or may not, at the controller's option, consider units that are inoperative. Detection of a duplicate unit number condition on one unit of a multi-unit drive is treated as a duplicate unit number condition on all other units that share one or more of the following components with the unit having the duplicate unit number:

a. A unit number select mechanism.

b. A Run/Stop or Load/Unload switch.

c. A spindle or other mechanical components.

2. Discovery of a duplicate unit number condition does not affect any unit that is already "Unit-Online". A unit that is already "Unit-Online" remains in that state until some other event (such as an AVAILABLE command or loss of controller context) occurs that would otherwise cause it to become "Unit-Available", at which time the unit becomes "Unit-Offline" and is spun-down as described below. Note, however, that such a unit is "Unit-Offline" to all other hosts (and therefore cannot be brought "Unit-Online" by them) even while it remains "Unit-Online" to its current hosts.

3. When a controller becomes aware of a duplicate unit number condition on two or more units connected to it, it immediately spins-down all such units that are "Unit-Available". When a unit that was "Unit-Online" with a duplicate unit number condition becomes "Unit-Available" for any reason, the controller immediately spins it down. In both cases, units that belong to a multi-unit drive and share a spindle or other mechanical components are to be spun down only if all of the units of the drive are "Unit-Offline" or "Unit-Available".

4. The controller returns "Unit-Offline" as the state for all units with duplicate unit number conditions that are not already "Unit-Online".

5. The controller must recognize when a duplicate unit number condition goes away. That is, the controller must recognize when all units except one with the same duplicate unit number have their unit numbers changed and/or become unknown. When this occurs, the controller must send AVAILABLE attention messages for the remaining unit (since it has just become "Unit-Available").

In addition to the above algorithms, controllers report duplicate unit number conditions to class drivers using the DUPLICATE UNIT NUMBER attention message. This allows hosts to complain to a human operator, who will presumably remedy the condition. This message is sent to all "Controller-Online" class drivers that have attention messages enabled when a duplicate unit number condition is first detected. It is permissable for a controller to send redundant or extra DUPLICATE UNIT NUMBER attention messages, provided that the reported duplicate unit number condition actually exists. See Section "Duplicate Unit Number Attention Message" for more details.

The above algorithms effectively enforce non-duplicate unit numbers across the drives connected to the same controller. Although not required by MSCP, it is recommended that class drivers use the following algorithm to enforce non-duplicate unit numbers across the drives accessible to that class driver:

1. Upon becoming aware of a unit (i.e., upon receiving an AVAILABLE attention message), the class driver should check if it is already aware of a unit with the same unit number on a different MSCP server. If it is not aware of such a unit, it should exit. If it is aware of such a unit, it should check if that unit has the same unit identifier.

2. If the unit identifiers are the same, it is the same unit multi-accessed to several controllers, so exit. If the unit identifiers are different, the class driver should issue a GET UNIT STATUS command to see if the old unit still exists.

3. If the old unit no longer exists (i.e., it's "Unit-Offline"), exit. If the old unit still exists, the class driver should check that the unit identifier returned by GET UNIT STATUS is also different from the unit identifier that was in the AVAILABLE attention message. If the unit identifiers are the same, exit. If the unit identifiers are different, the class driver should issue an AVAILABLE command with the "Spin-down" modifier set for the new unit. The class driver should also issue an AVAILABLE command with the "Spin-down" modifier set for the old unit if the class driver is not already "Unit-Online" to that unit.

4. Whenever a class driver brings an MSCP server "Controller-Online", the class driver should issue, through that MSCP server, AVAILABLE commands with the "Spin-down" modifier set for all units that it has "Unit-Online" via another MSCP server.

5. Whenever a class driver brings a unit "Unit-Online", the class driver should issue ONLINE commands for that same unit number to all MSCP servers. If more than one ONLINE commands succeeds, the class driver should check that the unit identifiers returned by all the successful ONLINE commands are the same. If any of the unit identifiers are different, then the class driver should treat all the ONLINE commands as having failed and issue AVAILABLE commands with the "Spin-down" modifier set to all MSCP servers on which the ONLINE command succeeded.

Note that this algorithm uses the fact that an AVAILABLE command with the "Spin-down" modifier set suppresses AVAILABLE attention messages until a human operator interacts with the unit.

4.5 Command Catagories and Execution Order

MSCP commands fall into one of four command categories. Each category has a set of execution order restrictions that MSCP servers must satisfy. The four categories and their restrictions are described below.

Immediate commands are those commands, such as status inquiries, that both require very little time to complete and do not cause any unit context changes. MSCP servers must process immediate commands immediately, without waiting for any other commands to complete. MSCP servers must guarantee that all outstanding immediate commands plus an additional GET COMMAND STATUS command issued by each "Controller-Online" class driver will complete within the controller timeout interval.

Class drivers may issue Immediate commands whenever their credit balance is greater than zero, whereas all other commands may only be issued when the class driver's credit balance is two or larger. This is discussed further in Section "Class Driver / MSCP Server Communications". Class drivers are thus guaranteed to be able to issue at least one Immediate command, and have it executed, regardless of what other commands they may have outstanding. In particular, the class driver is guaranteed to be able to issue a GET COMMAND STATUS command and have it complete within the controller timeout interval.

Sequential commands are those commands that, for the same unit, must be executed in precise order. Sequential commands typically alter a unit's context, such as by changing a unit characteristic. All sequential commands for a particular unit that are received on the same connection must be executed in the exact order that the MSCP server receives them. The execution of a sequential command may not be interleaved with the execution of any other sequential or non-sequential commands for the same unit. Furthermore, any non-sequential commands received before and on the same connection as a particular sequential command

must be completed before execution of that sequential command begins, and any non-sequential commands received after and on the same connection as a particular sequential command must not begin execution until after that sequential command is completed. Sequentional commands are, in effect, a barrier that non-sequential commands cannot pass or penetrate.

Non-sequential commands are those commands that controllers may re-order so as to optimize performance. Controllers may furthermore interleave the execution of several non-sequential commands among themselves, performing each command a fragment at a time. The only restrictions are:

1. Execution of a non-sequential command must not cross the barrier created by a sequential command for the same unit.

2. The controller must complete useful work on its oldest outstanding command within the controller timeout interval, so as to not cause a command timeout (see Section "Command Timeouts").

Special commands are similar to non-sequential commands, but have additional, unique execution order requirements. The execution order requirements for special commands are described in the commands' description.

Execution order is based on the order in which commands are received by the controller's MSCP server. For commands sent by a single class driver, the order of transmission is identical to the order of reception. For commands sent by several class drivers, the order of reception is essentially random. The only way that a class driver can ensure that one of its commands will be received after some other command issued by another class driver is to wait until the other class driver receives the first command's end message (i.e., wait until the first command completes) before sending the command.

For the purpose of determining execution order, a command is completed when the controller's MSCP server queues the command's end message for transmission to the host. The controller need not wait until the host has confirmed its reception, although sequential message delivery guarantees must be preserved. The gist of this is that, after termination of the connection between a host class driver and an MSCP server, the absence of a sequential command's end message implies nothing about the execution or non-execution of the sequential command.

4.6 Class Driver / MSCP Server Synchronization

Synchronization of a class driver with an MSCP server is accomplished by establishing or re-establishing the connection between the class driver and the MSCP server. When the connection is established or re-established, the MSCP server aborts or otherwise terminates all commands that are outstanding from that class driver. This forces the dialogue between the class driver and MSCP server to a known, synchronized state; namely that of having no outstanding commands. After establishing the connection the class driver can issue commands without worrying about duplicating command reference numbers or other unfortunate side effects. Note that synchronizing with the MSCP server, if successful, causes the MSCP server to become "Controller-Online".

As stated above, the main purpose of synchronization is to guarantee that there are no outstanding commands, thus forcing the dialogue between the class driver and MSCP server to a known state. MSCP servers must ensure that this guarantee is met before they allow synchronization to complete (i.e., before they become "Controller-Online"). In particular, MSCP servers must guarantee that no end messages will be sent and that no units will have their state, context, or (data) contents changed for any commands that were issued on an earlier incarnation of the connection between the class driver and MSCP server. Note that an MSCP server may allow outstanding commands to complete, either partially or entirely, but if it does it must delay the completion of synchronization (delay the transition to "Controller-Online") until all such commands have completed.

Class drivers must synchronize with the MSCP server whenever the host boots, recovers from a power failure, loses context, or is recovering from certain errors. After synchronizing with an MSCP server, the class driver should do the following:

1. Issue a SET CONTROLLER CHARACTERISTICS command to establish host settable controller characteristics and obtain non-host settable controller characteristics.

2. Issue ONLINE commands for all units that the class driver wishes to be "Unit-Online".

3. Re-issue all commands, if any, that were outstanding before the class driver synchronized with the MSCP server in the exact order that they were originally issued. However, any commands that have been aborted (i.e., for which an ABORT command has been issued) must not be re-issued; instead the class driver should assume that the command has been successfully aborted, and is therefore no longer outstanding.

There is one exception to re-issuing all commands that were outstanding. The class driver must maintain a retry limit count,

to ensure that its oldest outstanding command won't be retried infinitely many times. The magnitude of this limit is host dependent, although the oldest command must be retried at least once. When the class driver's oldest outstanding command exceeds the retry limit count, it must be aborted and an error returned; all other outstanding commands, however, should still be retried. See Section "Command Timeouts" for more information.

It may be possible that a class driver will receive messages from an MSCP server after the class driver has initiated synchronization with the MSCP server but before the synchronization completes. Whether or not this is in fact possible is communications mechanism dependent, as it depends upon the detailed design of the communications mechanism and port driver. Any attention messages that the class driver receives during this interval should be discarded. Any error log messages should be logged in the normal fashion. Any end messages that the class driver receives during this interval may be either handled normally (thus completing the corresponding command) or else discarded. If the class driver discards one end message, it must discard all subsequent end messages until the synchronization completes. It is recommended, although not required, that class drivers handle all such end messages normally.

4.7 Class Driver Error Recovery

The principle method of error recovery used by class drivers is to re-synchronize with the MSCP server, as described in the preceding section. All communications mechanism failures and many controller failures are reported by terminating the connection between the class driver and MSCP server, in response to which the class driver should attempt to re-synchronize with the MSCP server. If the class driver decides that the controller is insane, either because the class driver received an invalid message or because a command timed out, it should recover by re-synchronizing with the MSCP server. Similarly, if the MSCP server decides that the class driver is insane, it may terminate the connection to the class driver. If the class driver is in fact actually sane, it will re-synchronize with the MSCP server after the port driver notifies it that the connection has been terminated.

Aside from re-synchronization as described in the preceding paragraph, class drivers need perform very little error recovery, since controllers handle all recoverable errors. The only exceptions to this guideline are as follows:

1. Errors on multi-access drives should be retried using another controller.

2. Commands that fail due to the unit being "Unit-Available" should typically be re-issued after bringing the unit "Unit-Online".

3. High availability systems may wish to perform the enhanced error recovery described below.

High availability systems may wish to use the following additional error recovery strategy, in order to minimize the impact of certain types of controller problems. Rather than re-issuing outstanding commands all at once after re-synchronizing with a controller, such a system should instead re-issue the oldest outstanding command all by itself, preferably with the "Express Request" command modifier set. The host class driver should re-issue all other outstanding commands only after the oldest command completes. The other outstanding commands may be re-issued either all at once (recommended) or one at a time.

If the oldest command times out or otherwise fails again after being re-issued, then it was presumably the source of the problem and normal operation will resume. If the oldest command succeeds, then the problem is most likely due to some race condition within the controller and will not re-occur. If the problem does re-occur, then successive iterations of this algorithm will execute commands one at a time until the offending command is found and discarded.

4.8 This section deliberately omitted.

4.9 Host Access Timeouts

MSCP servers provide host access timeouts to guarantee that multi-access drives will be accessible in spite of certain communications mechanism failures. If the communications path between hosts and a controller fails when one or more drives are "Unit-Online" via that controller, the drives would normally become inaccessible. The drives can't be accessed via the controller through which they are "Unit-Online", since that controller can't be accessed, and they can't be accessed via a second controller, since they are "Unit-Online" through the first controller. The host access timeout, if enabled, eliminates this problem by causing the first controller to automatically release all drives if it doesn't receive a command within a specified time interval.

The exact mechanism used for host access timeouts is to have the controller's MSCP server become "Controller-Available" relative to any host class driver whose host access timeout expires. The MSCP server automatically resets the timeout whenever it receives a command from the class driver or has a command outstanding from that class driver; therefore the timeout will never expire if

the class driver maintains a reasonably constant dialog with the MSCP server. Note that implementing host access timeouts on a per host basis has the benefit that the MSCP server will automatically release any resources allocated to a host if that host goes down.

If communication with all hosts ceases, the host access timeout of each class driver will ultimately expire, causing the MSCP server to become "Controller-Available" relative to all hosts. Each unit will be released, allowing it to be accessed via an alternate controller, as soon as all class drivers that are "Unit-Online" with respect to the unit (or any other unit that shares its access path) cease to be "Unit-Online" by virtue of becoming "Controller-Available". Ultimately all class drivers will become "Controller-Available", thus releasing all units.

Class drivers specify the time interval that an MSCP server will use for the host access timeout in the SET CONTROLLER CHARACTERISTICS command. A default host access timeout of 60 seconds is used from the time a class driver becomes "Controller-Online" until it issues its first SET CONTROLLER CHARACTERISTICS command.

Each class driver may specify a separate time interval. This allows class drivers to vary the host access timeout interval in accordance with host policy and availability requirements. High availability systems should typically use a fairly short host access timeout, on the order of 10 to 30 seconds, together with a background process that verifies the continued availability of the host to controller communications path. The background process would have the desirable side effect of ensuring that the host access timeout never expired. Normal systems should use a larger timeout, on the order of 1 to 5 minutes, to avoid excessive resynchronizations, yet still allow failure recovery. Single user and other specialized systems may disable host access timeouts. Note that host access timeouts should typically be enabled even on systems that do not seem to require them, as drives may be multi-accessed to a totally separate host used as a backup system.

MSCP servers must implement host access timeouts subject to the following constraints:

1. The host access timeout expires, for a particular class driver, at the amount of the host access timeout interval after the controller or MSCP server completes all outstanding commands from that class driver, provided that no new commands are received from that class driver during this interval.

2. The MSCP server must enter the "Controller-Available" state (as a result of host access timeout expiration) relative to a class driver no sooner than the moment of host access timeout expiration defined in item 1 above.

3. The MSCP server should enter the "Controller-Available" state (as a result of host access timeout expiration) as soon as possible after the moment of host access timeout expiration defined in item 1 above. Except when the controller is saturated with work for other class drivers, this must be no later than one second plus the amount of the host access timeout interval after the moment of host access timeout expiration.

In other words, the host access timeout is measured from the time that the last command is completed. The MSCP server may defer recognition of host access timeout expiration for up to one second plus the amount of the host access timeout after the formal expiration of the timeout. That is, the host access timeout must be implemented with an accuracy range of -Ø% through +1ØØ%+1 second. Furthermore, this accuracy range may be extended on the high side if the controller is saturated with work for other class drivers.

This definition of host access timeouts has been structured to allow at least two alternative implementations. In the first implementation, the controller or MSCP server starts a timer when it goes "idle" -- when it completes the last outstanding command. The duration of the timer is the host access timeout interval. The timer must be designed to not err on the low side (too short an interval) and to err by no more than a factor of two on the high side (too long an interval). If the timer expires before the MSCP server receives another command, declare a host access timeout and become "Controller-Available". The second implementation uses a continuously running timer which "ticks" at the host access timeout interval. If the timer "ticks" twice in succession without there being any outstanding commands or without the MSCP server having received a command from the host, then declare a host access timeout and become "Controller-Available".

Each controller or MSCP server has a minimum and a maximum host access timeout interval that it implements. If a class driver specifies a host access timeout interval that is less than the minimum, then the MSCP server uses its minimum. If a class driver specifies a host access timeout interval that is greater than the controller's maximum, then the controller uses its maximum. A controller's or MSCP server's minimum host access timeout interval must be 1Ø seconds or less. A controller's or MSCP server's maximum host access timeout interval must be at least 255 seconds. That is, all controllers must fully implement host access timeout intervals in the range 1Ø through 255 seconds inclusive. Support of intervals outside this range is controller dependent. The range of host access timeout intervals that a controller supports must be described in the controller's Functional Specification; it cannot be obtained via MSCP.

Note that all controllers and MSCP servers must also implement the host access timeout value zero, which disables host access timeouts.

4.10 Command Timeouts

Host class drivers use command timeouts to guarantee that all controller or communications mechanism failures will be detected. The failures detected by command timeouts include partially sane or deadlocked controllers, which may continue to process new commands even though one or more old commands have been lost and will never complete. The use of command timeouts centers around the GET COMMAND STATUS command; indeed, the primary purpose of the GET COMMAND STATUS command is for command timeouts.

A controller is sane if and only if it will ultimately complete all commands submitted to it. For practical purposes, the term "ultimately" must be replaced with the phrase "within reasonable time". What constitutes a "reasonable time" varies with the complexity of the command and the performance of the controller and drives. We can eliminate the difficulty of the host class driver having to derive this "reasonable time" by re-stating the definition of a sane controller as follows: A controller is sane if and only if it will always complete useful work on its oldest outstanding request within reasonable time. This definition lets us set the "reasonable time" to some fixed value, and vary the units in which we measure "useful work" according to the complexity of the command. Command timeouts are based on this second definition.

A class driver implements the command timeout mechanism as follows. For each MSCP server to which it is "Controller-Online", the class driver keeps track of which command is its oldest outstanding command plus the previous "command status" value for its oldest outstanding command. The previous "command status" value should be set to all ones whenever the oldest command completes and a new command becomes the oldest. The class driver should issue GET COMMAND STATUS commands for its oldest outstanding command at intervals of the controller timeout interval or longer. When each GET COMMAND STATUS command completes, the class driver must check if the command for which it was issued is still outstanding and, if it is, verify that the "command status" returned by the GET COMMAND STATUS command is lower than the previous "command status"; this value measures the amount of work remaining before completion of the command. If the value ever increases or stays the same, or if a GET COMMAND STATUS command ever takes longer than the controller timeout interval to complete, then the class driver should assume that the controller has failed and re-synchronize with the controller.

In addition to guaranteeing that they will complete useful work on their oldest outstanding command, controllers must also guarantee that they will complete all aborted commands within the controller timeout interval. That is, an aborted command's end message must be sent no later than the amount of the controller timeout interval after the ABORT command's end message. Host class drivers can check this by setting the previous "command status" value to one whenever the oldest outstanding command has been aborted (i.e., when the class driver receives the ABORT command's end message indicating that its oldest outstanding command has been aborted).

Single host controllers -- i.e., controllers that do not provide Multi-Host support -- need not guarantee that all aborted commands will complete within the controller timeout interval if excessively pathological situations arise. Examples of such situations include:

1. ACCESS or ERASE commands whose byte counts exceed the maximum data transfer size imposed on READ and WRITE commands by the communications mechanism. For example, the Unibus imposes an architectural maximum transfer size of 2**18 bytes, since that is the size of the Unibus address space. Therefore ACCESS and ERASE commands whose byte counts exceed 2**18 bytes are excessively pathological for controllers that use the Unibus as their communications mechanism, so the controller need not meet the abort timeout requirements when such commands are outstanding.

2. Compound or multiple errors, causing error recovery sequences to stretch out to unrealistic lengths.

If the command timeout for an aborted command expires due to such a situation, the class driver will re-synchronize with the MSCP server and re-issue all outstanding commands except for those that had been aborted. Since the aborted commands will not be re-issued, the timeout will not re-occur. Thus this exception is transparent to host class drivers.

The above algorithm applies to non-immediate commands. With one exception the same algorithm may also be used for immediate commands, although a simpler one (using the fact that all immediate commands complete within the controller timeout interval) is also appropriate. The one exception is the GET COMMAND STATUS command used by the timeout algorithm itself. This command must be timed out as follows. If the class driver's credit balance is zero when it attempts to issue the GET COMMAND STATUS command, it must queue the GET COMMAND STATUS command for immediate transmission (before any other commands that may be outstanding) when its credit balance becomes non-zero. If the controller timeout interval expires again before the GET COMMAND STATUS command has both been transmitted and completed, then the class driver should assume that the controller has failed. Note

that the class driver's credit balance is guaranteed to be non-zero when all outstanding immediate commands have completed. Note also that controllers or MSCP servers must guarantee that all outstanding immediate commands plus one additional GET COMMAND STATUS command will complete within the controller timeout interval.

Upon concluding that a controller has failed, the class driver must re-synchronize with the controller's MSCP server and re-issue all commands (except commands that it has tried to abort) that were outstanding to that MSCP server in the same order that they were originally issued. In particular, the oldest outstanding command -- the one that timed out -- must be issued first (after initial SET CONTROLLER CHARACTERISTICS and ONLINE commands). The only exception to this is if the command's retry count expires. The class driver should maintain a retry count of the number of times the oldest command has timed out and been retried, and abort the command if the retry count exceeds a host dependent limit. The size of this retry limit is determined by host policy, except that all commands must be retried at least once. Note that sub-code zero of the "Controller Error" status code has been reserved for commands that exceed their retry count. This sub-code must never be generated by MSCP servers; it is generated by class drivers as a standard means of reporting command timeout errors.

In order to implement command timeouts, the host class driver must first obtain the controller timeout interval via the SET CONTROLLER CHARACTERISTICS command. Therefore the class driver should issue a SET CONTROLLER CHARACTERISTICS command as the first command after becoming "Controller-Online". The class driver should use a controller timeout interval of 10 seconds for this initial command. The class driver must never use a time interval that is shorter than the controller specified controller timeout interval for its command timeout determination, although the class driver may use a time interval that is longer than the one specified by the controller. The controller timeout interval specified by the controller must not be larger than 4 minutes and 15 seconds (i.e., 255 seconds).

One characteristic of this command timeout algorithm is that MSCP servers need not implement, and indeed most will not implement, the GET COMMAND STATUS command for any command that the MSCP server can guarantee will complete within the controller timeout interval. The GET COMMAND STATUS command should always return the value zero as the "command status" of such a command. It is acceptable if, due to vagaries of controller optimization algorithms, such a command will occasionally timeout. This is acceptable, so long as the frequency of such timeouts is extremely small and the controller immediately begins processing the first command it receives after re-synchronization. Since the class driver re-issues commands in the same order that they were originally issued, the oldest or timed out command is re-issued first, effectively guaranteeing that it will not be

delayed again by the controller's optimization algorithms. (The simplest way for most controllers to implement this will typically be to treat the first transfer command received across a newly established connection as if it were an Express Request).

4.11 Disk Geometry and Format

The host accessible portion of a disk unit consists of a vector of fixed length blocks, called logical blocks. Logical blocks are identified by logical block numbers (LBNs) which range from zero through N-1 inclusive, where "N" is the total number of logical blocks on the unit. The logical blocks on a unit are divided into two mutually exclusive regions:

1. The host area consists of those logical blocks available for host data storage. Logical blocks in the host area have LBNs in the range zero throught US-1 inclusive, where "US" is the "unit size", or number of logical blocks in the host area. The host obtains "US" or the "unit size" from the ONLINE or SET UNIT CHARACTERISTICS command end messages.

2. The unit's Replacement and Caching Table (RCT), used to record bad block replacement and miscellaneous housekeeping information. This information is further described below. The RCT (actually, multiple copies of the RCT) occupies the logical blocks numbered US through N-1 inclusive.

All of the logical blocks in the host area are guaranteed to be "good" -- i.e., to be free of permanent or hard errors (media defects). Most controllers implement this via bad block replacement. A pool of replacement blocks, identified by replacement block numbers (RBNs), is provided on the disk. Replacement blocks are not directly accessible to hosts. All host area logical blocks that are bad -- i.e., that contain media defects leading to hard errors or large numbers of correctable errors -- are replaced by a replacement block. The mechanism used to perform this replacement, and to revector accesses to bad logical blocks to the proper replacement blocks, is described in the DEC Standard Disk Format and/or the controller's Functional Specification. The pool of replacement blocks is typically distributed throughout the disk, so that revectoring of logical block accesses to replacement blocks has negligible impact on performance. See Section "Bad Block Replacement" and DEC Standard Disk Format for more information on bad block replacement.

The logical blocks that contain the RCT are not guaranteed to be "good". Since the RCT describes the bad logical block to replacement block mapping, mapping bad RCT blocks to replacement blocks would present an insoluble recursion problem. Instead of using replacement blocks for bad RCT blocks, the RCT region actually contains multiple copies of the RCT. Hosts obtain the size of each RCT copy and the number of RCT copies via the GET UNIT STATUS command. The last copy of the RCT may be truncated. See the DEC Standard Disk Format. Note that the disk is unusable if the corresponding block is bad in every copy of the RCT.

The detailed format and access algorithms for the RCT are described in the DEC Standard Disk format. In general, however, each copy of the RCT contains the following information:

1. One entry per replacement block, identifying the logical block, if any, that has been replaced by the replacement block.

2. Context information identifying the bad block replacement operation, if any, that is currently in progress on this unit. This information is used to complete the bad block replacement operation if the host and/or controller should crash in the middle of bad block replacement.

3. A Volume Write Protect flag.

Alternatively, a controller may use a non-standard replacement scheme or some other scheme than bad block replacement to guarantee that the logical blocks in the host area are "good". The mechanisms used by such a controller to guarantee that the host area logical blocks are "good" must be totally invisible to hosts, and must not require host cooperation for their initialization, use, or maintenance. Such a controller or unit must still provide the first block of the RCT, which contains the Volume Write Protect flag.

The host visible portion of each logical block consists of a fixed number of data bytes. The number of data bytes is either 512 or 576, determined when the disk volume is formatted. All logical blocks on a disk volume have the same number of data bytes.

The data bytes are the only portion of logical blocks that are directly host visible. Certain other items, however, must be in each logical block as their presence is implied by various MSCP functions:

1. Each block must contain a forced error indicator, so as to properly implement and recognize the "Force Error" command modifier.

2. Each block must contain a bad block indicator, so that references to bad blocks can be efficiently revectored to the proper replacement blocks. This is unnecessary if the controller does not use bad block replacement to provide a "perfect" host area.

As stated above, the host area of a disk is structured as a vector of logical blocks. From a performance viewpoint, however, it is more appropriate to view the host area as a four dimensional hyper-cube, the four dimensions being cylinder, group, track, and sector. Thus, it is possible to decompose a logical block number into a unique quadruple of numbers, namely the block's cylinder number, group number, track number, and sector number. Cylinder number is most significant and sector number is least significant.

Alternatively, we can define a track as consisting of a fixed number of blocks, a group as consisting of a fixed number of tracks, and a cylinder as consisting of a fixed number of groups. The position of a block within a track is the block's sector.

The terms sector, track, and cylinder all come from the geometry of classical disk drives. Groups can be viewed as an optimization for short seeks whose seek time is easily predictable.

At any particular instant, the set of logical blocks that are potentially accessible is those blocks in all tracks that are in the same sector, group, and cylinder. In the absence of transfers to a different group or cylinder, this set of potentially accessible blocks changes over time by keeping the group and cylinder constant while incrementing the sector (modulo the number of blocks/sectors in a track). Referring to our hyper-cube analogy, the set of potentially accessible blocks form a line parallel to the track axis. This line moves parallel to the sector axis, wrapping around when it reaches the edge of the hyper-cube. A disk therefore provides cyclic access to the blocks in a particular group and cylinder.

This access structure to logical blocks implies that, to a close approximation, the track to which a block belongs has no effect upon performance. That is, switching tracks within the same group and cylinder effectively requires zero time. Two separate transfers in the same group and cylinder and for the same sectors have similar performance, regardless of what tracks they are on. To a first order approximation, two separate transfers on successive sectors of different tracks in the same group and cylinder have the same performance as a single, two sector (block) transfer.

Changing cylinders, however, does require a certain amount of time. The amount of time required to switch between two cylinders is approximated by a monotonically increasing function of the difference between the two cylinder numbers. The time to switch between cylinders is typically not affected by whether or not groups are also being changed. After switching cylinders, the sector position of the disk (i.e., which sector's blocks are immediately accessible) is unpredictable.

Changing groups also requires a certain amount of time. Generally, the time to switch between groups in the same cylinder is no more than, and often less than, the time to switch between one cylinder and the next. If cylinders and groups are both changed at the same time, the time to switch groups is effectively zero, as it is included in the time to switch cylinders.

When changing from one group to the next (successive) group within the same cylinder, the time required to switch groups is predictable, so that transfers are optimized. If a transfer to sector S in group G is followed by a transfer in group G+1 of the same cylinder, then sector S+1 (modulo the number of sectors/blocks in a track) is the optimal sector for the new transfer and sector S is the maximally unoptimal sector for the new transfer. The main effect of this is to optimize continuous (spiral) transfers that cross group boundaries.

Note that what has been described herein is the model for disk logical geometry, which may have a tenuous relationship to a disk's actual physical geometry. Disk designers should devise a logical to physical geometry mapping which optimizes the accuracy of the model herein described. This will generally be done as follows. Head or track switches that effectively require zero time (i.e., that require less than the inter-sector time) will be reported as logical MSCP tracks. Head or track switches that require significant amounts of time will be divided into two classes: those that require only a little time (typically less than one rotation) and whose time is predictable, and those that require somewhat more time and/or a lot of time. The switches that require only a little time will be reported as logical MSCP groups. The switches that require more time will be reported as logical MSCP cylinders, where the switches should be mapped to cylinders in such a manner as to minimize the amount of time required to switch between cylinders that are (numerically) close together.

The affect of this geometry on multi-block transfers is as follows. A multi-block transfer requires a certain minimum time, which is the time to transfer one block or sector times the number of blocks in the transfer. Crossing track boundaries requires no additional time. Crossing a group boundary requires a small, relatively fixed additional amount of time; this time is typically less than the time to transfer an entire track (i.e., less than one rotation). Crossing a cylinder boundary

requires a somewhat larger additional amount of time; this time is typically at least the time to transfer an entire track (i.e., at least one rotation).

The affect of this geometry on host allocation policies for random-access files is as follows. Whenever possible, a random-access file should be allocated within a single group. If this is not possible, the host should try to allocate it within a single cylinder. If this is also not possible, the host should allocate it in the minimum number of adjacent cylinders.

When a block has a high probability of being accessed immediately after another block, hosts should attempt to allocate both blocks in the same group or, if that is not possible, in the same cylinder. If both blocks cannot be allocated within the same cylinder, then they should be in cylinders that are as close together as possible.

Host's obtain the size of a unit's tracks, groups, and cylinders from the GET UNIT STATUS command's end message. Some of these disk geometry concepts may not apply to all disk units. Units report the fact that a concept doesn't apply by specifying its size to be the same as the next larger concept. For example, a disk unit that doesn't have groups would specify one group per cylinder, implying that groups and cylinders are the same size. Zero should be supplied for the cylinder size if the cylinder concept is inappropriate to a disk unit, which hosts should interpret as the entire unit being a single cylinder. The value zero may equivalently be interpreted as specifying an arbitrarily large number of groups per cylinder. This may propagate downwards; if both cylinders and groups are inappropriate to the unit, then zero would be provided for both their sizes. If the model of cyclic access to the sectors in a track is inappropriate, then the unit should typically specify one block per track.

The following examples illustrate how inappropriate concepts should be specified:

1. A "disk" implemented as a pure random-access memory (i.e., semiconductor RAMs) would specify one block per track (since cyclic access doesn't apply) and zero for the cylinder and group size (since the entire unit is effectively a single group).

2. A classical DECtape could be specified several ways, but one block per track, one track per group, and one group per cylinder (i.e., each block is a separate cylinder) is probably the most natural.

3. A single loop shift register or delay line (such as an ultrasonic delay line) would specify zero for the track, group, and cylinder size, since the entire unit is effectively a single track.

There is one exception to the above discussion, concerning the specification of track size. Track size, in addition to being useful for performance prediction, is also used in the bad block replacement algorithm specified in DEC Standard Disk Format. It is possible that the track size appropriate for performance considerations will be different from the track size required by bad block replacement. If this occurs, the track size required by bad block replacement must be specified, as the performance effects are a secondary consideration.

Note that all of this performance oriented disk geometry only applies to the host area of a disk unit. The concepts need not apply to the Replacement and Caching Table (RCT) and any other areas of a disk, as access to those areas is not performance sensitive and/or not host visible.

4.12 Bad Block Replacement

Bad block replacement is a technique used with disk class devices to present each unit as a single logically contiguous set of usable blocks. See the preceding section for a discussion of logical blocks, replacement blocks, and a high level description of bad block replacement.

Most bad blocks are detected when a disk volume is manufactured. These blocks are always replaced when the volume is formatted, as are any other blocks that the formatter can determine are bad. Other blocks become bad during normal use, and must be replaced dynamically. MSCP is solely concerned with dynamic bad block replacement.

Usually, but not necessarily always, the host performs bad block replacement in response to the controller reporting a bad block. The algorithm used to perform bad block replacement is described in DEC Standard Disk Format.

Controllers only report bad blocks (to hosts) in transfer command end messages. This is a direct consequence of the fact that controllers only detect bad blocks while performing disk transfers. They report bad blocks to hosts by means of the "Bad Block Reported" end message flag. If this flag is set, then the "First Bad Block" field in the end message is the logical block number of the first bad block (lowest block number) encountered by the transfer. A second flag, the "Bad Blocks Unreported" end message flag, is set to indicate that multiple bad blocks were encountered by the transfer, implying that one or more were not reported to the host. After replacing the first bad block, the host may (at its option) reissue the transfer to determine the next bad block, repeating this process until all of the bad blocks are replaced. Given the likely incidence of multiple bad blocks in a transfer, it is unclear that this yields any benefit.

When performing bad block replacement, the host must access and update the unit's Replacement and Caching Table (RCT), inform the controller that the block has been replaced, and initialize the new replacement block. The host accesses and updates the RCT using ordinary transfer operations, specifying a logical block number in the range assigned to the RCT. The host informs the controller that the block has been replaced with the REPLACE command; this allows the controller to reformat the disk to reflect the bad block replacement. The new replacement block is initialized with a normal WRITE command, specifying the bad block's logical block number, after the REPLACE command has completed.

While a bad block replacement operation is in progress, the details of the replacement operation being performed are recorded in a portion of the unit's Replacement and Caching Table (RCT). The information recorded includes the bad block's LBN, the replacement block's RBN, and the data to be written into the block at the conclusion of the replacement operation. Recording this information in the RCT allows the bad block replacement operation to be successfully completed in the event of a system crash, power failure, or other interruption. When the unit next becomes "Unit-Online", the host must check the RCT and complete any replacement operation that was in progress. At the same time the host must check the Volume Write Protect flag and take the actions described in Section "Write Protection".

4.13 Write Protection

There are several ways that a unit or volume may be write protected under MSCP:

Hardware Write Protection

> The unit's write protect mechanism has been activated by a user, causing the unit to be write protected.

Software Write Protection

> The host has requested that the unit be write protected.

When Hardware Write Protection is established (i.e., when a user activates the unit's write protect mechanism), the controller must provide a smooth transition to the write protect state. That is, the controller must complete all write operations (commands) that it has already initiated on the unit before actually prohibiting writes. Note, however, that the controller should immediately reject any new write operations that it receives after the user activates the unit's write protect mechanism. Write operations that the controller received before the write protect mechanism was activated, but that haven't been

initiated yet, may either be rejected or completed at the controller's option. The end result of this is that each individual write command or operation is either completed in its entirety or else rejected before any of its data is written to the unit. Note that this issue cannot arise with Software Write Protection, as they are established and cleared by sequential commands, which implies that no write operations can be outstanding.

Note that it is not possible to perform bad block replacement when a unit is Hardware Write Protected.

A unit's Write Protect Status Display Mechanism must indicate the "inclusive or" of all forms of write protection. That is, the display must indicate that the unit is write protected both when it is Hardware Write Protected and when it is Software Write Protected.

Whenever a host brings a unit "Unit-Online", it must check the "Volume Write Protect" flag in the unit's RCT. This is in addition to the other checks it must make for a partially completed bad block replacement operation (see Section "Bad Block Replacement"). If the RCT flag is set, the host should immediately (software) write protect the unit with a SET UNIT CHARACTERISTICS command. When a user or a higher level host process subsequently decides to write enable the volume, the Software Write Protect status must be cleared with a SET UNIT CHARACTERISTICS command and the RCT accessed to clear the flag. See DEC Standard Disk Format for RCT access algorithms and the detailed format of these RCT flags.

4.14 Compare Operations

MSCP includes the following kinds of compare operations:

1. The COMPARE HOST DATA command.

2. Read-compare operations, invoked by the "Compare Reads" unit flag or by the "compare" modifier on a READ command.

3. Write-compare operations, invoked by the "Compare Writes" unit flag or by the "compare" modifier on a WRITE command.

The operation of these different types of compare operations is described below. Note that all of the compare operations report the first difference or other error starting from the beginning of the transfer. Therefore the compare operation at the end of the transfer may be aborted if a difference is discovered at the beginning.

The COMPARE HOST DATA command is used to verify that data in host memory matches data on a unit. The data is obtained from the unit in the manner that is most convenient or efficient for the controller. In this respect the COMPARE HOST DATA command operates identically to a READ command. Unlike the READ command, the data is not transferred to host memory; instead, data is obtained from host memory and compared against the data obtained from the unit. Upon completion of the command the controller reports whether the data was identical or different. The data being different is reported as a "Compare Error" in the command's end message. However, no error log message is generated as this is not considered to be a "significant" error (since it can be deliberately caused by user programs).

Read-compare and write-compare operations are performed at the conclusion of the appropriate transfer commands to verify that the data was correctly transferred and that the data can now be obtained from its destination. The general algorithm used is to obtain the data from its destination and compare it against the data re-obtained from its source.

If a read-compare or write-compare operation fails, the controller must interpret this as implying that the original transfer failed and therefore retry the original transfer if appropriate. If the controller successfully obtains the data from its source and destination, but the data is different, then the controller must retry the original transfer and report the compare error in an error log message. If the controller cannot successfully obtain the data from its destination, but the error is one that may be eliminated by re-writing the data to its destination, then the controller must also retry the original transfer and report the error (from the attempt to obtain the data from its destination) in an error log message. All other errors need not be retried, but must be reported in an error log message. The only exception to the above is commands that have the "Suppress Error Recovery" modifier set; the controller may or may not, at the controller's option, retry the original transfer if a compare error occurs in such a command.

For example, "Data Errors", such as an uncorrectable ECC error, must be retried on write-compare operations. They need not be retried on read-compare operations, since an unrecoverable "Data Error" implies that the READ itself will fail. "Compare Errors" must always be retried. Note that the controller need not discriminate amoung types of errors -- it may always retry all errors during read-compare or write-compare operations, regardless of whether or not the error will inhibit the original transfer.

The number of retries required for read-compare and write-compare operations is controller dependent. However, all controllers must retry such operations at least once. The exact number of retries that a controller implements should be chosen based on undetected error rate characteristics. The controller may either

retry the entire transfer, or else only retry the portion that includes the error.

4.15 Multi-Unit Drives and Formatters

A multi-unit drive is a single physical disk drive that appears as several independent units to hosts. So-called fixed plus removable disk drives, providing one removable disk unit and one non-removable disk unit, are the most common example of multi-unit drives. A multi-unit formatter is a single set of interface or read/write electronics that connects several otherwise independent units to controllers.

All the units of a multi-unit drive or formatter share a single access path to controllers. This implies that all of the units must be "connected" to the same controller. In particular, if one unit of a multi-unit drive or formatter is "Unit-Online" via a controller, then all the other units of the multi-unit drive or formatter may only be accessed by that same controller. That is, the other units are "Unit-Offline" to all other controllers. Awareness of this characteristic is critical for high availability systems -- if a failed operation on a multi-access unit is to be retried via another controller, and the unit is part of a multi-unit drive or formatter, then all units of the drive or formatter must be switched to the other controller.

Some units of a multi-unit disk drive may share mechanical components as well as interface electronics. Such units are said to share a spindle. That is, the units must either be all spinning or all not spinning, just as if the units shared a drive motor or spindle (which they typically will). Such units must also share a single Run/Stop switch, since they are always spun-up and spun-down together. Hosts must also be aware of units which share a spindle, as dismounting one such unit requires that all units sharing the same spindle be spun-down.

The units of a multi-unit drive or formatter are identified by the "multi-unit code" unit characteristic field. Hosts obtain this two byte field via the AVAILABLE attention message or in the end message of a GET UNIT STATUS, ONLINE, or SET UNIT CHARACTERISTICS command. The low byte of this field contains a controller dependent encoding of the access path between the controller and the drive. The high byte of this field contains a controller dependent encoding of the spindle, on a particular access path, that the unit uses. Controllers may use any encoding whatsoever, provided that each access path and each spindle (within an access path) has a unique value. Note that the access path byte is implicitly qualified by the controller's or MSCP server's identity, and that the spindle byte is implicitly qualified by the access path.

Hosts use the "multi-unit code" field as follows. When a host decides to spin-down a unit, it scans all other units that are "Unit-Online" via the same MSCP server for those units whose entire "multi-unit code" field (both bytes) matches the unit being spun-down. Such units, if any, share a spindle or other mechanical components with the unit being spun-down, so that they must be spun-down together. When a host decides to access a unit via a different controller, it scans all other units that are "Unit-Online" via the same MSCP server for those units whose low byte of the "multi-unit code" field matches the unit being switched. Such units, if any, share an access path with the unit being switched, so that they must also be switched to the new controller.

Note that the low byte of the "multi-unit code" field (the access path) is meaningless for units that are inherently restricted to a single controller. Controllers may return any fixed value as the access path encoding for such units, provided that it doesn't duplicate the value returned for any units on the same controller that are not inherently restricted to a single controller.

This use and format of the "multi-unit code" field implies the following architectural restrictions on all controllers:

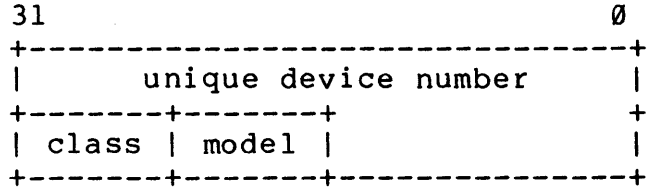
1. All units that share a spindle or other mechanical components must also share an access path. Note that this is an essential restriction for multi-unit drives, regardless of how shared components are communicated. If units that share a spindle did not share an access path, then they could be simultaneously "Unit-Online" via different controllers, making it impossible to coordinate a simultaneous spin-down.

2. There is a maximum of 256 access paths per controller. In the absence of multi-unit drives or formatters, this implies a maximum of 256 units per controller.

3. There is a maximum of 256 spindles per access path. In the absence of shared spindles, this implies a maximum of 256 units per access path or formatter.

4.16 Controller and Unit Identifiers

MSCP requires that all controllers and drives have unique identifiers, called controller identifiers and unit identifiers. The structure of these identifiers is as follows:



The "class" byte identifies the type of the subsystem -- controller, disk drive, etc. The "model" byte identifies the exact model of the subsystem within its class. All valid class and model codes are non-zero, implying that all valid identifiers are non-zero. The "unique device number" field must uniquely identify the device among all devices of that same class and model. The device serial number could be used as the "unique device number", although that isn't required. Currently defined values for the "class" and "model" bytes are listed in Appendix C. Values for new devices must be added to that appendix, via an ECO to this specification, as new products are developed. Note that different units of multi-unit drives are distinguished by having different "model" bytes; the "class" and "unique device number" fields are typically identical. Note also that all MSCP servers for the same device class within the same controller must return the same controller identifier.

As previously stated, MSCP requires that controller and unit identifiers be unique across all devices accessible via MSCP. This clearly cannot be checked by controllers. Controllers can, however, enforce unique unit identifiers across the units that are attached to themselves. This is done using the following algorithms:

1. Controllers should detect and respond to duplicate unit identifiers across all units whose unit identifiers the controller can obtain, including all units that would otherwise be "Unit-Online" or "Unit-Available". Detection of a duplicate unit identifier on one unit of a multi-unit drive is treated as a duplicate unit identifier condition on all other units that share one or more of the following components with the unit having the duplicate unit identifier:

   a. A unit number select mechanism.

   b. A Run/Stop or Load/Unload switch.

c. A spindle or other mechanical components.

Note that duplicate unit identifiers are detected regardless of the state of a unit's Run/Stop or Load/Unload switch.

2. Whenever a controller becomes aware of a duplicate unit identifier, it immediately spins-down all units with the duplicate identifier and forces them to remain spun-down. The controller spins-down the units regardless of their current state. The controller typically forces them to remain spun-down by spinning them down again whenever an operator spins them up.

3. The controller returns "Unit-Offline" with sub-state "inoperative" as the state for all units with duplicate unit identifiers. In addition, if the unit might potentially be connected to another controller, the controller should flag the presence of the duplicate unit identifier in the drive. Other controllers, if any, must check this flag and also treat the unit as "Unit-Offline" if the flag is set.

Whether or not a controller does in fact check for duplicate unit identifiers is controller dependent. Note that a duplicate unit identifier is a drastic failure, indicative of some otherwise undiagnosed hardware malfunction.

4.17 Media Type Identifiers

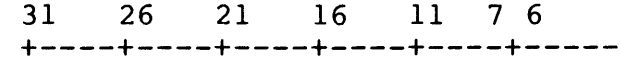
Controllers return a media type identifier for each unit accessible via that controller. This identifier encodes two pieces of information:

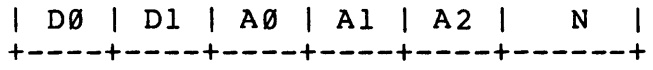
1. The preferred device type name for use with the unit. These two alphabetic characters are conventionally used with the unit number and a controller designator as the fully qualified operating system identifer for the unit.

2. The name (product name) of the media used on the unit. This name should be printed on the unit's front panel and on all removable media that may be used with the unit.

The primary reason for returning this information is to simplify operating system support for generic device allocation.

The media type identifier returned via MSCP is a 32 bit quantity encoded as follows:





where the fields are as follows:

DØ
Dl The preferred device type name for the unit. DØ and Dl are five bit fields, each encoding one alphabetic character. "A" is encoded with the value 1, "B" with the value 2, etc. DØ encodes the left character of the device type name, Dl the right character.

AØ
Al
A2
N The name of the media used on the unit. AØ through A2 are five bit fields, each encoding an alphabetic character or null. "A" is encoded with the value 1, "B" with the value 2, etc. Zero represents a null or the abscence of a character. One to three characters of the media name are encoded, left justified, in AØ through A2. N is a seven bit field containing the value of two decimal digits.

Note that the encoding of the media name assumes that the name consists of one, two, or three alphabetic characters followed by exactly two digits (i.e., ann, aann, or aaann). MSCP requires that the product names for all mass storage devices adhere to this format.

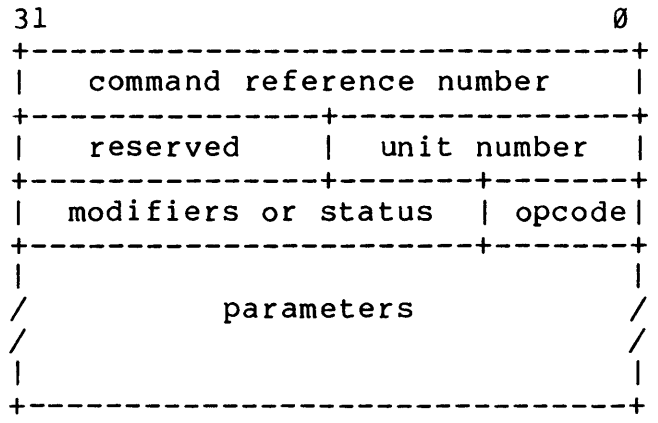
Currently defined device type and media names (i.e., currently defined media type identifier values) are listed in Appendix C. Names or values for new devices must be added to that appendix, via an ECO to this specification.

CHAPTER 5

MSCP CONTROL MESSAGE FORMATS

5.1 Generic Control Message Format

All MSCP control messages consist of a 12 byte header and a 36 byte or shorter parameter area. The device class (e.g., disk) does not appear in the control message; it is implied by the connection or MSCP server to which the control message is sent. Multi-byte numbers are stored least significant byte first (i.e., using the standard VAX11 number formats). Messages are laid out as follows:



The length of the parameter area varies depending upon the opcode.

The communications mechanism conveys both the text of a message and its length. The receiver of a message uses the its length to verify that all required parameters are in fact present.

The communications mechanism may restrict the allowable message lengths. For example, it might require that all messages have a fixed length of 48 bytes or that the length be an even multiple of 4 bytes. For this reason the message lengths defined by MSCP are minimum lengths; senders may pad messages as necessary to meet communications mechanism length restrictions. The contents of the padding -- that is, the contents of any data past the end of the message formats shown in this document -- are reserved and must follow the rules for reserved fields defined in the

following section. (i.e., such padding must contain zeros).

The fields in the message header are interpreted as follows:

command reference number

A 32 bit, unique, non-zero number used to identify host commands. Class drivers should supply a unique reference number in each command that they send to an MSCP server. The MSCP server copies the reference number to the command's end message and to all error log messages that relate to that specific command. The MSCP server supplies a reference number of zero in attention messages and in error log messages that do not relate to a specific host command. A class driver may supply a zero reference number if it does not need to associate a command with its end message.

Command reference numbers must be unique across all commands that are outstanding on the same connection. That is, they must be unique across all outstanding commands issued by a single class driver (host) to a single MSCP server. The class driver may re-use a command's reference number when the command is no longer outstanding -- i.e., after receiving the command's end message or after re-synchronizing with the MSCP server. Command reference numbers need not be unique for commands issued by different class drivers -- i.e., commands issued by different hosts or commands for different MSCP servers from the same host. Therefore controllers must internally use the combination of a command reference number and the connection on which the command was received as the unique identifier of an outstanding command.

Command reference numbers are not interpreted in any way by MSCP servers. Their purpose is to provide a unique identifier by which class drivers can name commands. They are used by class drivers to match end messages and error log messages with the corresponding command message and to identify the object of an ABORT or GET COMMAND STATUS COMMAND.

unit number

Identifies the specific unit within the device class to which the message applies. This value is the binary equivalent of the decimal unit number displayed by the unit select mechanism.

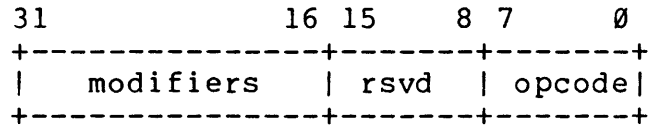
opcode

Identifies the meaning or purpose of the message. In messages sent from a class driver to an MSCP server, this field specifies the operation or command to be performed. In messages sent from the controller to the class driver, this field specifies whether this is an end message or an

attention message. The opcode of an end message also identifies the type (opcode) of the command to which the end message corresponds. A message's opcode implicitly specifies the length and format of the message, including the interpretation of any parameters that are present.

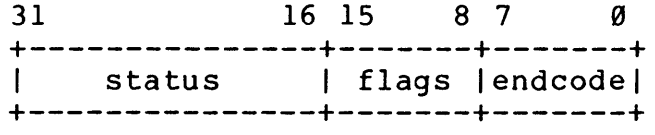
modifiers or status

This field has different formats in command messages and end messages, and is reserved in attention messages. In command messages this field has the following format:



The "modifiers" field contains bit flags that modify the operation identified by "opcode", or zero if no modifiers are specified.

In end messages this field has the following format:



The "status" field identifies the completion status of the command; the "flags" field contains bit flags, called end flags, that report certain conditions that are disjoint from normal completion status of a command. These fields are further described in Sections "End Message Format" and "Status Codes".

5.2 Reserved and Undefined Fields

Reserved fields are those fields that are intended for possible future extensions to MSCP. The use of such fields must follow certain rules, in order to ensure that such future extensions can be upwards compatible with the current version of MSCP. In general, the sender of a message must supply the value zero in all reserved fields. The action for a message receiver varies, and is discussed below.

An undefined field is just that -- its contents are controller implementation dependent, and therefore cannot be used in any meaningfull way by class drivers. Undefined fields are provided in order to simplify controller implementation. Class drivers must ignore the contents of undefined fields.

5.2 Reserved and Undefined Fields

A field, as used in this discussion, may have any length. In particular, it may be an individual bit of a flags word or byte as well as an entire byte, word, or whatever.

Class drivers must supply the value zero in the reserved fields of all messages (commands) that they send to a controller, and must also ignore the contents of reserved fields in all the messages (end messages, attention messages, and error log messages) that they receive from an MSCP server. MSCP servers must supply the value zero in the reserved fields of all messages (end messages, attention messages, and error log messages) that they send to class drivers. MSCP servers must either ignore the contents of reserved fields in the messages (commands) that they receive from class drivers or verify that the contents are zero; the command is treated as invalid if the contents are non-zero. Whether or not an MSCP server verifies that reserved fields are zero is controller dependent, and need not be consistent for all reserved fields.

Many controllers generate command end messages by simply modifying the commands' command messages. That is, the controller copies a command message into an internal buffer, modifies it in place during execution of the command, then sends the resulting contents of the internal buffer as the command's end message. To simplify such an implementation, controllers may merely "echo" command message reserved fields when the corresponding field in the end message should be zero. More precisely, if some field in the end message of a command should be zero, and the corresponding (same position) field in the command's command message is a reserved field, the controller may copy the reserved field from the command message to the end message rather than explicitly zeroing the field in the end message.

The above paragraphs have listed all of the allowable controller actions when a controller receives a command message with a non-zero value in a reserved field. That is, when a controller receives a command message with a non-zero value in a reserved field it must do one of the following:

1. Reject the command as invalid and return an Invalid Command end message.

2. Totally ignore the non-zero contents of the reserved field. That is, the command's execution, results, and end message contents are totally unaffected by the non-zero value.

3. If the corresponding field in the end message should have a zero value, echo the contents of the reserved field in the command message as the value of the field in the end message. In all other ways totally ignore the non-zero contents of the reserved field. That is, the command's execution, results, and the contents of

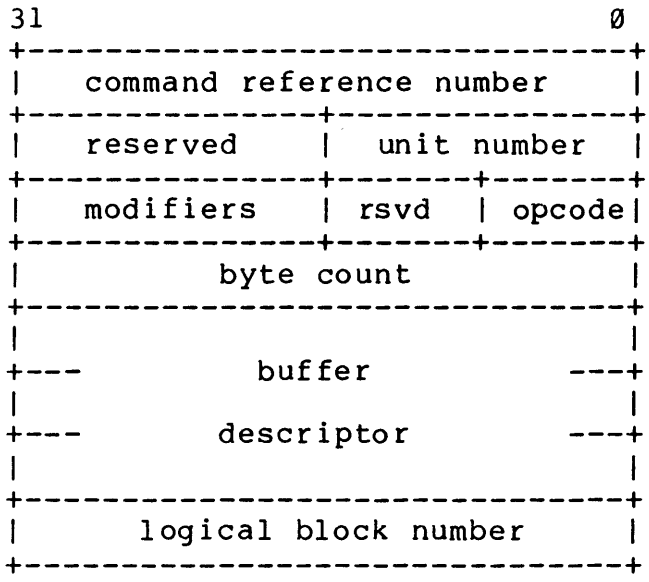
all other fields in the end message are totally unaffected by the non-zero value.

Note that option 3 is primarily of use when a field is reserved in both command and end messages.

Note that controllers must implement option 1 whenever the internal functioning of the controller may be altered if a reserved field contains a non-zero value.

5.3 Transfer Command Message Format

Although the parameters and their layout is command (opcode) dependent, many commands perform data transfers and thus use similar sets of parameters. Therefore data transfer related parameters are always laid out as follows:



where these parameters are interpreted as follows:

byte count

The total requested length of the data transfer in bytes. For disk class devices, the "byte count" must meet the requirements described in the next paragraph.

If the "logical block number" field identifies a logical block in the host area of the disk volume (i.e., the "logical block number" is less than the "unit size" returned in the ONLINE and SET UNIT CHARACTERISTICS end messages), then the "byte count" must be less than or equal to the following maximum byte count:

(unit size - logical block number) * block size

where "unit size" is the unit's host area size (returned in the ONLINE and SET UNIT CHARACTERISTICS end messages), "logical block number" is the contents of the "logical block number" field in the command message, and "block size" is the volume's block size, either 512 or 576 bytes. The controller or MSCP server must check that the "byte count" is less than or equal to the above maximum. That is, the controller or MSCP server must reject any transfer command that begins in the host area of a disk volume and attempts to continue into the volume's Replacement and Caching Table (RCT). An "Invalid Command" status code with an "Invalid Byte Count" sub-code must be returned if this restriction is violated.

If the "logical block number" field identifies a logical block in the disk volume's Replacement and Caching Table (RCT) (i.e., the "logical block number" is greater than or equal to the "unit size" returned in the ONLINE and SET UNIT CHARACTERISTICS end messages), then the "byte count" must be exactly the sector size (either 512 or 576 bytes). If a different "byte count" value is provided, the controller may either perform the transfer with the specified "byte count" or else return an "Invalid Command" status code with an "Invalid Byte Count" sub-code.

For all disk transfer commands that contain "buffer descriptors" (i.e., all transfer commands except ACCESS and ERASE), the "byte count" must also be less than or equal to the size of the buffer identified by "buffer descriptor". Note that "buffer descriptor", and thus the size of the buffer, is inherently communications mechanism dependent. The size of a buffer is not necessarily available to the MSCP server until it attempts to transfer past the end of the buffer. A "Host Buffer Access Error" status code is returned if the "byte count" exceeds the length of the buffer. Note that such errors are not necessarily distinguishable from other causes of "Host Buffer Access Errors".

For disk transfer commands only, some communications mechanisms may prohibit odd "byte count" values. A "Host Buffer Access Error" status code is returned if the "byte count" is an illegal odd byte count.

"Byte count" values that exceed any of the maximum values described above may be detected either before the transfer is initiated or when the transfer attempts to cross the boundary from legal to invalid byte counts. If detected before the transfer is initiated, the MSCP server must not transfer any data and must return zero in the "byte count" field of the end message. If detected when the transfer attempts to cross the boundary, the MSCP server must transfer all data up to the maximum legal byte count and return the maximum legal byte count in the "byte count" field of the end message; data must not be transferred past the maximum legal byte count. Which algorithm an MSCP server uses for detecting

byte counts that are too large is controller dependent.

buffer descriptor

Communication mechanism dependent identification of the host buffer to use for the data transfer. The information encoded in this 12 byte (96 bit) field includes:

- o A host identifier (port or node identification).
- o The name of a buffer on the host.

Note that the inclusion of a host identifier allows for third party transfers. The buffer descriptor formats used by various communication mechanisms are listed in Appendix D.

logical block number

The logical block number (position) on the disk volume at which to start the data transfer. This value must not identify a block past the end of the volume's Replacement and Caching Table (RCT). Section "Disk Geometry and Format" describes the mapping of logical block numbers to disk volume regions. This error causes the command to be rejected with an "Invalid Command" status code and an "Invalid Logical Block Number" sub-code.

5.4 Command Modifiers

The allowable modifiers on a command are command (opcode) dependent. The individual command descriptions list the allowable modifiers for each command. All modifiers that are not explicitly allowed for a command are reserved, and must be treated in accordance with the requirements for reserved fields described in Section "Reserved and Undefined Fields". Modifiers that are only allowed on one command are described in that command's description. Modifiers that are common to many commands are described below:

Compare

Applicable to data transfer commands. After the transfer, the data will be read back from the transfer destination and verified against the original data re-obtained from the source. Specifying this modifier is similar, but not identical, to following the transfer command with a COMPARE HOST DATA command. In particular, if the compare operation fails, an error log message is generated (if enabled) and the original transfer operation retried. (With the COMPARE HOST DATA command, an error log message must not be generated on compare errors and retries are unnecessary, although innocuous and therefore allowable). See Section "Compare

Operations", for a more detailed description of this modifier's effects.

Express Request

Applicable to non-sequential commands. This modifier requests that the controller ignore its normal optimization policies in order to complete this command as quickly as possible. The exact implementation of express requests is controller dependent -- in general the controller will complete some or all of its outstanding commands before completing an express request.

Use of express requests disables the normal controller guarantees that ensure that all commands are serviced in a timely manner. If express requests are repeatedly issued, some or all other outstanding commands may time out (i.e., never be completed).

Express requests do NOT override sequential command execution guarantees. Some controllers may completely ignore the express request modifier; the exact treatment of express requests should be described in a controller's Functional Specification.

Force Error

Applicable to write commands. Causes the data to be written with the forced error indicator set, so that all attempts to read the data will fail. The error will be preserved until the next time the block is written. The forced errors produced with this modifier must be recognized by the controller as deliberate and never reported to the error log.

Suppress Error Correction

Suppresses error correction mechanisms, such as ECC correction. Error recovery mechanisms, such as retries, are not affected by this modifier. This modifier, in effect, lowers the threshhold at which an error is considered to be uncorrectable. It typically lowers the threshhold to zero, although that is not required; since error correction is drive type dependent, the lowered error correction threshhold is also drive type dependent. If a drive has several error correction mechanisms, it is permissable for this modifier to suppress some and not affect others.

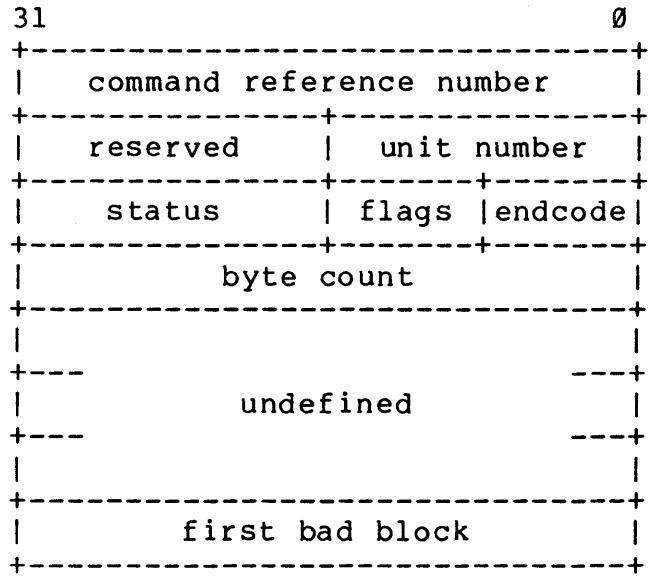
Suppress Error Recovery

Suppresses most error recovery mechanisms, such as read retries. Error correction mechanisms, such as ECC correction, and some error recovery mechanisms, such as seek retry, are not affected by this modifier. The exact definition of which error recovery mechanisms are suppressed

and which are not affected is drive type dependent.

5.5 End Message Format

An MSCP server sends an end message to a class driver to report completion of a command. The generic end message format is as follows:



The command reference number and unit number are copied from the command message. The remaining fields are as follows:

endcode

Identifies this message as an end message and the type of command (opcode) that this is an end message for. This field implicitly specifies the format and interpretation of the parameters.

flags

Bit flags, collectively called end flags, used to report various conditions detected due to this command but not directly related to success or failure. The following flags are defined:

Bad Block Reported

Set if one or more bad blocks were detected. Indicates that the host should replace the bad block identified in the "first bad block" field.

Bad Blocks Unreported

Set if one or more bad blocks were detected and not reported in the "first bad block" field. That is, two or more bad blocks were detected and the "first bad block" field only reports the first bad block in the transfer.

Error Log Generated

Set if one or more error log messages were generated that refer to this command -- i.e., that contain this command's command reference number. This flag allows the host to save any outstanding command context that it wishes to include in the error log. The MSCP server must send the error log messages either before or shortly after it sends the end message containing this flag.

All other bits in this field are reserved, and must be treated in accordance with the requirements for reserved fields described in Section "Reserved and Undefined Fields".

status

The modifiers field is used for a completion status code. The status code indicates whether the operation was successfully completed or, if it wasn't successful, what type of error occurred. Note that recoverable errors are reported as successful completion of the command. All errors, whether recoverable or not, are reported in a separate error log message if they should be logged.

If several errors occur in a transfer operation, the status code reports the first error starting from the beginning of the transfer (i.e., the lowest byte count or lowest logical block number). The only exception is transfer commands that include a compare operation (i.e., read-compare and write-compare operations); errors in the original transfer always take precedence over errors during the compare operation.

If a "Forced Error" and some other error occur at the same point (same byte count or logical block number) within a transfer operation, the other error must be reported. If a "Compare Error" and some other error which is not a "Forced Error" occur at the same point within a transfer operation, the other error must be reported. If a "Forced Error" and a "Compare Error" both occur at the same point, and no other error occurs at that point, the "Compare Error" must be reported. Otherwise, which error of multiple errors occurring at the same point should be reported is controller dependent. An alternative way of stating this is that "Forced Errors" are the error of last resort and that "Compare Errors" are the error of second to last resort.

If several errors occur in a non-transfer operation, the error that is reported is controller dependent unless the individual command description states otherwise.

byte count

In transfer command end messages, the number of bytes successfully transferred, counting from the start of the transfer to the first error (i.e., the lowest byte count or lowest logical block number with an error). Data that follows the first error is not counted, even if transferred successfully. The only exception is transfer commands that include a compare operation (i.e., read-compare and write-compare operations); errors in the original transfer always take precedence over errors during the compare operation.

The controller must have successfully transferred all data up to the point identified by "byte count". Furthermore, the error identified by "status" must have actually occurred at the position identified by "byte count". The state of the transfer following the position identified by "byte count" is undefined. None of the transfer following "byte count" may have been performed or attempted, all of it may have been attempted (with unknown success), or some parts may have been attempted and others not.

Again, the only exception to this is transfer commands that include a compare operation. Such a command makes two passes over the data; one for the original transfer and another for the compare operation. For most errors, there is no way to determine in which pass the error was detected. Therefore the only guarantee is that the original transfer was performed up to the point identified by "byte count" without detecting any errors; the compare operation may or may not have been performed up to that point. If a "Compare Error" is reported, then both the original transfer and the compare operation have been successfully performed up to the point identified by "byte count"; the state of both the original transfer and the compare operation after that point, however, is undefined. This implies that the compare pass of a transfer command that includes a compare operation may be done for the entire transfer as a unit, block by block, or anywhere in between.

For disk class devices, the granularity of the byte count on errors (i.e., the resolution with which the point of error is identified) need not be any smaller than the volume's block size. That is, the byte count need only identify the block in which the error occurred, rather than the exact word or byte. In particular, the byte count returned with such errors as "Compare Errors" and "Host Buffer Access Errors" need only identify the block in which the error occurred, rather than the exact word or byte. Note that a block is

identified by the number of the first byte in the block. Controllers may optionally provide finer granularity for the byte count field on errors. If an error is not reported, controllers must return the exact byte count that was in the command message.

Not present in non-transfer command end messages.

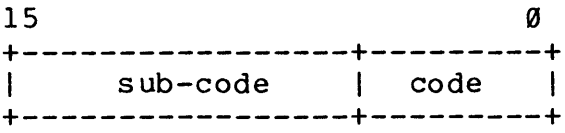
first bad block

In disk transfer command end messages, the logical block number of the first bad block (i.e., the bad block with the lowest logical block number) detected during the transfer that the host should replace. Only valid if the "Bad Block Reported" flag is set; undefined (garbage) if the "Bad Block Reported" flag is clear.

Not present in non-transfer command end messages.

5.6 Status Codes

The "status code" field is divided into a 5-bit major status code and an 11-bit status sub-code arranged as follows:



The "event code" field of error log messages has the identical structure and encoding. Errors that are reported in both an end message and an error log message use identical values for the "status code" and "event code" fields. The same value may not be used to report a different type of event as a status code than as an event code.

The 5-bit major status code conveys the status information that hosts need for normal operation. Therefore the major status codes are a formal part of MSCP. All controllers must return the same major status codes for similar situations.

The 11-bit sub-code exists to specify the exact error or unusual situation encountered with very fine detail. As such it is primarily used for diagnostic purposes, and hosts should not need to examine it during normal operation.

Sub-codes related to protocol or state errors are a formal part of MSCP. All controllers must return the same sub-codes for protocol or state errors. These sub-codes are generally bit flags, allowing several causes of the major status code to be reported.

Sub-codes related to controller and/or drive errors, however, must be allowed to vary from one controller or drive to another. There is no requirement that the same sub-codes be returned for similar drive or controller errors. These sub-codes are generally specific values, corresponding to one specific event or error. Each sub-code must have the same meaning whenever it is used. It is the use of a sub-code that may vary (i.e., whether or not a specific controller returns that sub-code), not its meaning. The defined sub-codes are listed in Appendix B; this list may expand (via an ECO to MSCP) whenever a new drive or controller type is introduced.

The major status codes that may be returned in end message "status code" fields are listed below along with the general use made of sub-codes. The actual sub-codes used are listed in Appendix B. Those sub-codes that are a formal part of MSCP are also listed in the descriptions of the commands that may return them.

Success

The command was successfully completed. This status code may also be returned, for some commands, if the intended effect of the command has already been accomplished (i.e., requesting a drive that isn't spinning to spin-down). The sub-code consists of bit flags used to report various "alternate" forms of success; see the individual command descriptions for details.

The status code value associated with "Success" is, by definition, zero. Sub-code value zero (i.e., no sub-code bits set) is "Normal" success, and implies normal completion of a command. One sub-code bit, the "Duplicate Unit Number" bit, is common to many commands. This bit, when set, implies that the unit is "Unit-Online" and the command succeeded, but that the unit has a duplicate unit number. The unit will become "Unit-Offline", due to the duplicate unit number, as soon as it ceases to be "Unit-Online". Other "Success" sub-code bits are unique to a particular command, and are described under the individual command descriptions.

Invalid Command

This status code is used for two purposes:

1. In normal command end messages, it is used to report invalid parameter values (e.g., bad logical block number). Some controllers may not detect certain invalid parameters until after performing some part of the command. For example, a byte count that runs past the end of a disk may not be detected until the transfer has been performed up to the end of the disk.

2. In the Invalid Command End Message, it is used to report invalid MSCP commands (protocol errors). A command is invalid if some field contains a reserved value or the command message was too short to contain all the parameters required by the opcode.

Note that an unknown unit number does not constitute an invalid parameter or command; unknown unit numbers are treated as if the unit is "Unit-Offline".

The sub-code is used to report the offset, within the command message, of the field in error. Bits 8 through 15 (the high byte) of the "status code" field contain the byte offset from the start of the message to the field in error. Multi-byte fields are identified by the offset to their lowest byte. Single byte fields positioned at an odd offset may be identified, at the controller's option, by either their actual offset or their offset minus one. That is, offsets may be truncated to an even value. Sub-code zero is used to report that the command message was too short to contain the parameters required by the command's opcode. Note that any value is valid for the field at offset zero, the "command reference number".

Note that protocol errors, reported via this status code and the Invalid Command end message, may cause the MSCP server to become "Controller-Available" relative to the class driver that issued the invalid command.

Command Aborted

The command was aborted by an ABORT command. The end message for the aborted command (i.e., the end message containing the "Command Aborted" status code) has the normal format for the command and all fields are valid. In particular, the "byte count" field identifies how far a transfer command was completed before it was aborted. The status of the transfer beyond the returned byte count is undefined. Sub-codes are not used.

Unit-Offline

The unit identified by the "unit number" field of the end message is in the "Unit-Offline" state. The sub-code consists of bit flags that indicate why the unit is "Unit-Offline". Note that there may be several reasons for the unit being "Unit-Offline". If the sub-code is zero, it implies that the unit is unknown -- i.e., the controller knows of no unit with the specified unit number.

Unit-Available

The unit identified by the "unit number" field of the end

message is in the "Unit-Available" state. The sub-code is always zero.

Media Format Error

Only returned by the ONLINE command for disk class devices. The volume mounted on the unit appears to be formatted incorrectly, so that it must be reformatted (and all data lost) before it may be used. This error is also returned if the volume is formatted with 576 byte sectors and the controller only supports 512 byte sectors. Note that the volume may only "appear" to be formatted incorrectly; the typical cause of this error is a fault in the drive's read/write electronics. If this is the case, the volume can usually be successfully accessed on another drive.

Controllers must treat the unit as if an AVAILABLE command with the "Spin-down" modifier set had been issued for it whenever they return this error code. The unit is therefore always in the "Unit-Available" state with AVAILABLE attention messages suppressed until a human operator changes the volume or spins-up the unit. The sub-code reports which integrity check the volume failed; it is volume format, and therefore drive type, dependent.

Write Protected

The unit identified by the "unit number" field of the end message is write protected and the command required that data be written onto the drive. The sub-code consists of bit flags indicating the reasons why the unit is write protected.

Compare Error

A COMPARE HOST DATA command, a read compare operation, or a write compare operation found different data in the host buffer and the unit identified by the "unit number" field of the end message. The sub-code is always zero.

Data Error

Invalid or uncorrectable data was obtained from a drive, as determined by internal error detecting or correcting codes. The sub-code is used to report the exact error detected. Sub-code zero is used for "Forced Errors". All errors caused by the "Force Error" modifier must be reported with sub-code zero.

Host Buffer Access Error

The controller encountered an error when attempting to access a buffer in host memory. The sub-code is used to report the exact error encountered. This status code is also returned whenever the command's buffer descriptor or byte count

violate any communications mechanism dependent restrictions.

Note that this status code is NOT used to report errors encountered when transferring command, end, attention, or error log messages between the controller and a host. Such errors are reported by terminating the connection between the class driver and MSCP server. The mechanism for reporting such errors to the host's error log is communications mechanism dependent.

Controller Error

The controller encountered an internal controller error. The sub-code is used to report the exact error encountered. An internal controller error is reported as a "Controller Error" if and only if the controller has reasonable grounds to trust its sanity and expects to complete, either successfully or with an appropriate error status code, all of its outstanding commands. All more severe controller errors are reported by terminating the connection between the controller's MSCP server and the host class driver. This is in addition, of course, to attempting to generate an error log message.

Sub-code zero of this status code is reserved for host detected command timeouts; all other sub-codes are controller dependent.

Note that some controller errors may be reported using other error codes, if an internal controller error causes the controller to mis-diagnose the error.

Drive Error

The controller discovered an error within a drive. Such errors are typically, but not always, mechanical in nature, since most non-mechanical errors are reported as "Data Errors". The sub-code is used to report the exact error encountered.

In many cases a "Drive Error" will indicate that the unit is broken or inoperative. If this occurs, the "Drive Error" should be reported once and the unit should subsequently be reported as being "Unit-Offline" due to being inoperative.

The status codes that may be returned for a specific command are command (opcode) dependent. The status codes that may be returned for each command and any special meaning that they have specific to the command are listed in the command descriptions. Note that the format of a command's end message is solely determined by its opcode; the status code returned in the end message does not affect the end message's format. The only two exceptions, protocol errors and serious exceptions, have unique end messages that contain a special opcode.

5.7 Unit Flags

Several messages contain a field called the unit flags field. This field consists of unit characteristics bit flags. Some unit flags are host settable; host settable unit flags may be set or cleared with the ONLINE and SET UNIT CHARACTERISTICS commands. Other unit flags are non-host settable; the controller must ignore the values supplied by hosts for such flags, and always return the correct value from the unit's characteristics. A few unit flags may be host settable or non-host settable, depending on the presence or absence of a command modifier.

Many unit flags, including all host settable unit flags, are only valid when the unit is "Unit-Online". The values returned for such unit flags are undefined if the unit is "Unit-Offline" or "Unit-Available". A few non-host settable unit flags are valid when the unit is "Unit-Available" and during certain "Unit-Offline" sub-states; these flags are identified in the individual flag descriptions below.

Those bits in the "unit flags" word that are not defined as unit flags are reserved, and must be treated in accordance with the requirements for reserved fields described in Section "Reserved and Undefined Fields".

The unit characteristics flags are as follows:

Compare Reads

> A host settable characteristic; set if all read transfers should be verified with a compare operation. Equivalent to specifying the "Compare" modifier on all READ commands. Undefined when the unit is either "Unit-Available" or "Unit-Offline".

Compare Writes

> A host settable characteristic; set if all write transfers should be verified with a compare operation. Equivalent to specifying the "Compare" modifier on all WRITE commands. Undefined when the unit is either "Unit-Available" or "Unit-Offline".

Removable Media

> A non-host settable characteristic; set if unit has removable media. Valid whenever the controller can determine the unit's characteristics; see the descriptions of the GET UNIT STATUS, ONLINE, and SET UNIT CHARACTERISTICS commands for more information.

Write Protect (hardware)

A non-host settable characteristic; set if and only if the unit's write protect mechanism is activated, causing the unit to be Hardware Write Protected. All write operations, including attempts to perform bad block replacement or otherwise modify the RCT, will be rejected when this flag is set. See Section "Write Protection". Undefined when the unit is either "Unit-Available" or "Unit-Offline".

Write Protect (software)

Normally a non-host settable characteristic; a host settable characteristic if the "Enable Set Write Protect" command modifier is asserted in the ONLINE or SET UNIT CHARACTERISTICS commands. Set if and only if the unit is Software Write Protected. See Section "Write Protection". Undefined when the unit is either "Unit-Available" or "Unit-Offline".

576 Byte Sectors

Normally a non-host settable characteristic; a host settable characteristic if the controller supports 576 byte sectors and the "Ignore Media Format Error" command modifier is asserted in the ONLINE command. Set if the volume mounted on the unit has 576 byte sectors. Undefined when the unit is either "Unit-Available" or "Unit-Offline".

5.8 Controller Flags

The SET CONTROLLER CHARACTERISTICS command is used to set and clear host settable controller flags, and to obtain the values of non-host settable controller flags. Host settable controller flags are stored on a per class driver basis; each class driver may have different settings for host settable controller flags. Non-host settable controller flags are fixed controller characteristics, and therefore common to all class drivers of the same device class. The controller must ignore the values supplied by hosts for non-host settable controller flags, and always return the correct value from the controller's characteristics.

All host settable controller flags are, by default, clear whenever a class driver becomes "Controller-Online" to an MSCP server. The flags remain clear until the class driver sets them with a SET CONTROLLER CHARACTERISTICS command or until the class driver is no longer "Controller-Online" to the MSCP server.

Those bits in the "controller flags" word that are not defined as controller flags are reserved, and must be treated in accordance with the requirements for reserved fields described in Section "Reserved and Undefined Fields".

The controller flags are as follows:

Enable Attention Messages

A host settable controller characteristic; set if attention messages should be sent to this host. Note that this flag is applicable to all attention messages, regardless of type.

Enable Miscellaneous Error Log Messages

A host settable controller characteristic; set if error log messages that do not relate to a specific command should be sent to this host.

Enable Other Hosts' Error Log Messages

A host settable controller characteristic; set if error log messages that relate to commands issued by other hosts should be sent to this host.

Enable This Host's Error Log Messages

A host settable controller characteristic; set if error log messages that relate to commands issued by this host should be sent to this host.

576 Byte Sectors

A non-host settable controller characteristic; set if the controller supports disks formatted with 576 byte sectors. Note that this flag is only applicable to the disk device class.

CHAPTER 6

MINIMAL DISK MSCP SUBSET

This section first describes the unit and controller flags that are used with the minimal Disk MSCP subset, then it describes each command to which controllers must respond, and finally it describes the attention messages that the controller must generate. The controller must respond with the Invalid Command end message and an "Invalid Opcode" status code to any command that is not listed here.

Each command description includes the command's category or execution order (see Section "Command Category and Execution Order"), the command message format, a list of the allowable command modifiers, the command's end message format, a list of possible status codes, and a description of the command's effects. Use of any command modifiers other than the ones listed for an individual command is reserved, and must be treated in accordance with the requirements for reserved fields described in Section "Reserved and Undefined Fields".

6.1 This section deliberately omitted

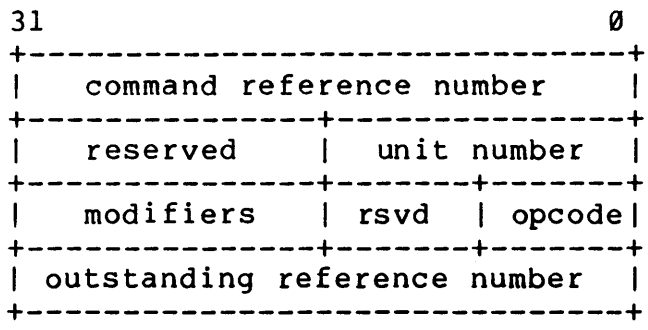
6.2 This section deliberately omitted

6.3 ABORT Command

Command category:

Immediate

Command message format:



unit number

Must be the same as the "unit number" field in the outstanding command to be aborted. This allows controllers to optimize their search for the outstanding command. If the incorrect unit number is supplied, some controllers may erroneously conclude that the command is no longer outstanding and therefore not abort the command.

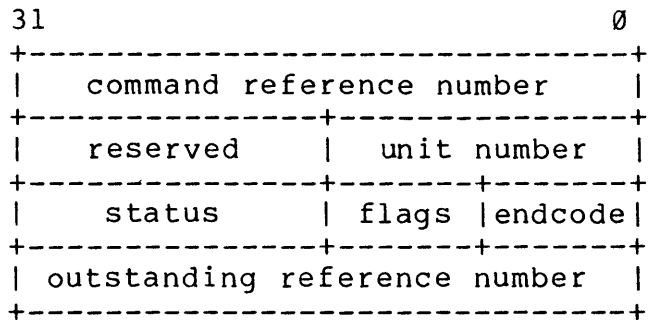
outstanding reference number

Command reference number of the command to be aborted. Note that a class driver may only abort commands that it itself has issued. This derives from the fact that command reference numbers are implicitly qualified by the connection on which the command was issued, so that class drivers have no way of naming commands issued by a different class driver.

Allowable modifiers:

none

End message format:



outstanding reference number

The command reference number of the command that was aborted. Identical to the value supplied in the ABORT command message.

Status Codes:

Success (sub-code "Normal")

Description:

The ABORT command causes a specified command to be aborted at the earliest time convenient for the controller. The specified command must, however, either be aborted or be completed within the controller timeout interval. See Section "Command Timeouts" for a discussion of the interaction between ABORT and command timeouts.

The ABORT command always succeeds; if the command to be aborted is not known to the controller, this implies that it has already completed and the ABORT command will be ignored. The controller always returns the "Normal" status code in the ABORT command's end message.

The controller may ignore the ABORT command if the command being aborted will always complete within the controller timeout interval.

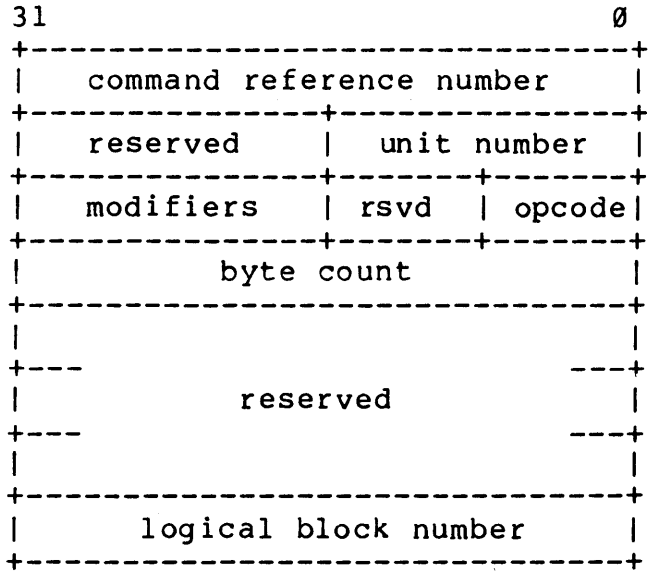
The class driver must wait for the aborted command's end message, or else re-synchronize with the MSCP server, before reusing the aborted command's command reference number or releasing the aborted command's context. If the command was aborted, its end message will contain the "Command Aborted" status code; otherwise the command was completed. The class driver may ignore the ABORT command's end message. Note that the class driver may receive the ABORT command's end message either before or after the aborted command's end message.

6.4 ACCESS Command

Command category:

Non-sequential

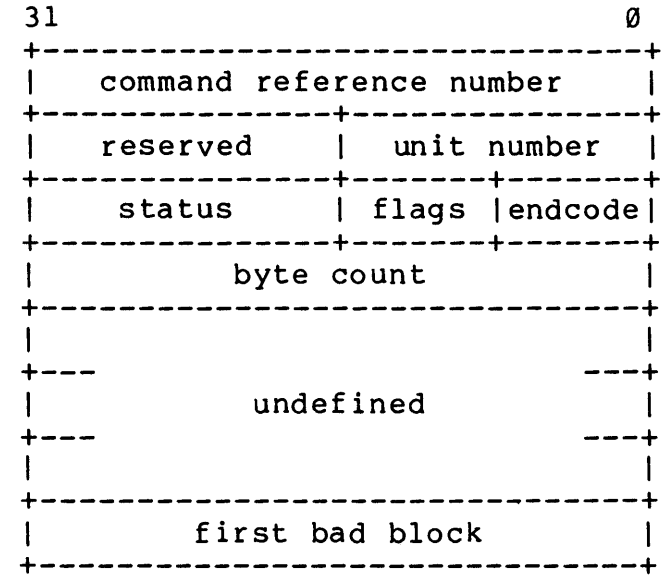
Command message format:



Allowable modifiers:

Express Request
Suppress Error Correction
Suppress Error Recovery

End message format:



Status Codes:

Success (sub-code "Normal")
Success (sub-code "Duplicate Unit Number")
Invalid Command (sub-code "Invalid Byte Count")
Invalid Command (sub-code "Invalid Logical Block Number")
Command Aborted
Unit-Offline
Unit-Available
Data Error
Controller Error
Drive Error

Description:

Data is read from the unit, checked for errors, and discarded. The purpose of this command is to verify that the designated data can be accessed (read) without error.

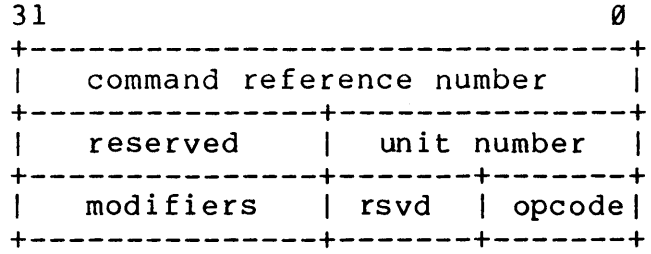
This command is exactly equivalent in function, although not in performance, to a READ whose data is ignored by the host.

6.5 AVAILABLE Command

Command category:

Sequential

Command message format:

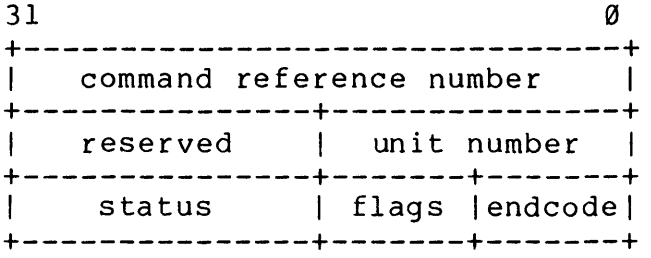


Allowable modifiers:

Spin-down

Requests that the disk stop spinning and that the heads be unloaded. Note that the command completes as soon as the spin-down has been initiated, rather than waiting for the disk to stop spinning. The spin-down will not be initiated if this unit belongs to a multi-unit drive and this unit shares a spindle with some other unit that is "Unit-Online"; see Section "Multi-Unit Drives and Formatters". Regardless of whether or not the spin-down is actually initiated, AVAILABLE attention messages will be suppressed for this unit, both for this controller and for any other controllers to which the unit may be connected; see Section "Unit States".

End message format:



Status Codes:

Success (sub-code "Normal")
Success (sub-code "Duplicate Unit Number")

Success (sub-code "Spin-down Ignored")

The "Spin-down Ignored" sub-code bit flag is set if and only if the "Spin-down" modifier was specified and one or more other units with which this unit shares a spindle is still "Unit-Online", preventing this unit from spinning down. See Section "Multi-unit Drives and Formatters" for an explanation of shared spindles.

Success (sub-code "Still Connected")

The "Still Connected" sub-code bit flag is set if and only if this unit may potentially be accessed via another controller and one or more other units with which this unit shares an access path is still "Unit-Online", preventing this unit from being accessed via the other controller (if any). The "Still Connected" sub-code bit flag will always be set if the "Spin-down Ignored" bit flag is set. See Section "Multi-access Drives" for a discussion of access paths.

Command Aborted

The unit's state has not changed.

Unit-Offline
Controller Error
Drive Error

Description:

All outstanding commands for the specified unit are completed, then the unit becomes "Unit-Available". If the "Spin-down" modifier was not specified, the unit is not already "Unit-Available", and no other units that share this unit's access path are "Unit-Online" (i.e., the "Still Connected" status sub-code bit flag is clear), then an AVAILABLE attention message is sent by any other controller to which the unit is connected. The controller to which this command was sent need not itself send an AVAILABLE attention message.

If the "Spin-down" modifier is specified, the disk spins down and its heads are unloaded, unless some other unit with which this unit shares a spindle is "Unit-Online". The disk may be spun up with an ONLINE command or by operator intervention. The "Spin-down" modifier also suppresses AVAILABLE attention messages for this unit, both for this controller and any

other controllers to which the unit may be connected. See Section "Unit States" for a discussion of suppressing AVAILABLE attention messages.

This command will be accepted if the unit is "Unit-Online" or "Unit-Available". It is nugatory to issue this command to a unit that is "Unit-Available" unless the "Spin-down" modifier is specified. Assuming no other errors occur, the "Success" status code will be returned regardless of whether the unit was previously "Unit-Online" or "Unit-Available".

If the unit was "Unit-Online" but had a duplicate unit number prior to the AVAILABLE command being issued, the AVAILABLE command may complete, at the controller's option, with either a "Success" or a "Unit-Offline" status code. The "Unit-Offline" status code must have the "Duplicate Unit Number" sub-code flag set. The "Success" status code may or may not, at the controller's option, have the "Duplicate Unit Number" sub-code flag set. Subsequent attempts to access the unit will return "Unit-Offline" status with the "Duplicate Unit Number" sub-code flag set unless the duplicate unit number has been eliminated.

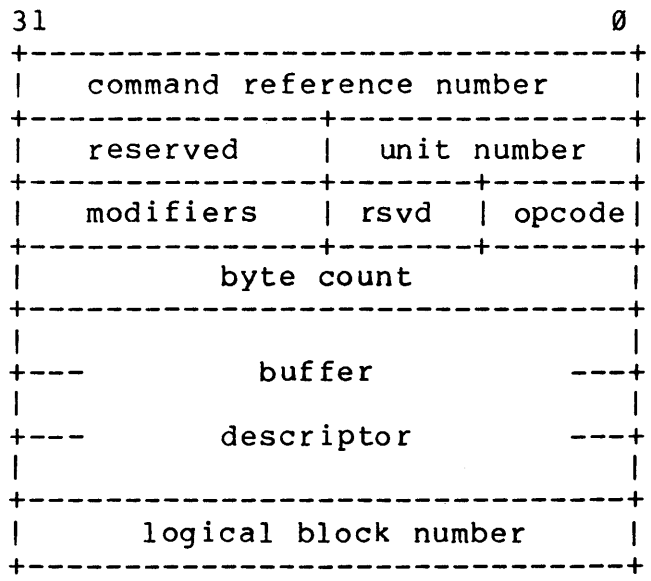
6.6 This section deliberately omitted

6.7 COMPARE HOST DATA Command

Command category:

Non-sequential

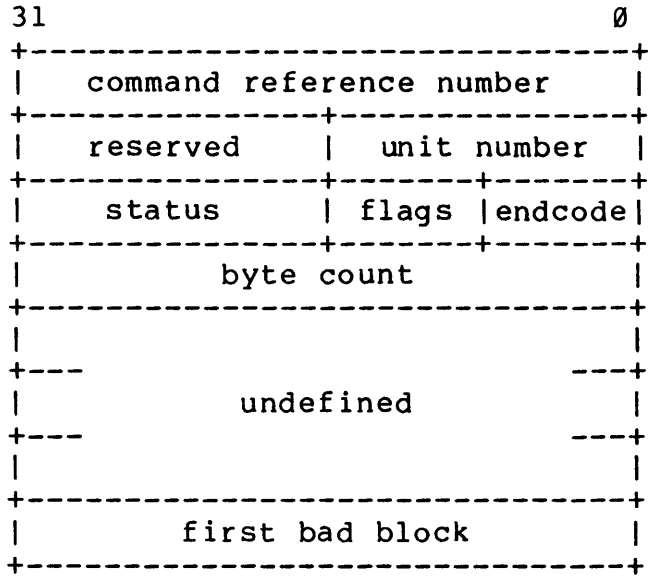
Command message format:



Allowable modifiers:

Express Request
Suppress Error Correction
Suppress Error Recovery

End message format:



Status Codes:

Success (sub-code "Normal")
Success (sub-code "Duplicate Unit Number")
Invalid Command (sub-code "Invalid Byte Count")
Invalid Command (sub-code "Invalid Logical Block Number")
Command Aborted
Unit-Offline
Unit-Available
Compare Error
Data Error
Host Buffer Access Error
Controller Error
Drive Error

Description:

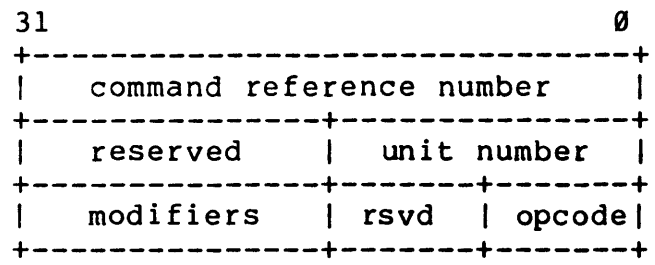
Data is read from the unit and compared with the data in the host buffer. A "Compare Error" is reported unless the data is identical. Note that the occurence of any other error, except a "Forced Error", at the same point as the "Compare Error" will override the "Compare Error". Note also that any "Compare Errors" detected by this command must NOT be reported to the error log.

6.8 DETERMINE ACCESS PATHS Command

Command Category:

see below

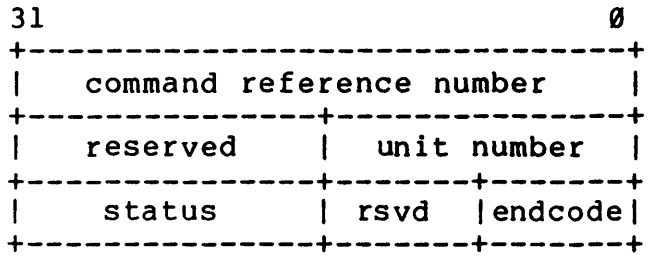
Command message format



Allowable modifiers:

none

End message format:



Status codes:

Success (sub-code "Normal")
Success (sub-code "Duplicate Unit Number")
Command Aborted
Unit-Offline
Unit-Available
Controller Error
Drive Error

Description:

Class drivers use this command to determine the topology of multi-access drive configurations. When sent to a unit that is "Unit-Online", it causes that unit and any other units that share its access path to identify themselves to any other controller to which they are connected. The MSCP servers in the other controllers will, as a result, become aware that the unit is online via the controller receiving this command. They will then send an ACCESS PATH attention message to their "Controller-Online" class drivers, thus informing the class drivers of the alternate access paths to the unit. This command must be treated as a no-op that always succeeds if the unit is incapable of being connected to more than one controller.

The actual notification to another controller, and thus the sending of an ACCESS PATH attention message, is dependent on the proper functioning of the unit and both controllers. Furthermore, it need not be 100% reliable. That is, assuming the unit and both controllers are functioning properly, there need only be a high probability (better than 50%) that the other controller will in fact be notified and send an ACCESS PATH attention message. For this reason, plus the fact that the topology may change while the unit remains "Unit-Online", hosts that need to know multi-access drive topology must issue DETERMINE ACCESS PATHS commands to all "Unit-Online" units on a periodic basis, such as once every 5 to 15 minutes.

This command in no way affects the unit's actual state to any controller. The unit remains "Unit-Online" via the receiving controller and remains "Unit-Offline" via other controllers. Note, however, that this command may affect the unit's "Unit-Offline" sub-state that is perceived by other controllers.

The processing of this command may, and often will, cause a transient performance degradation for the specified unit. Controllers must consider this performance degradation when specifying their controller timeout interval.

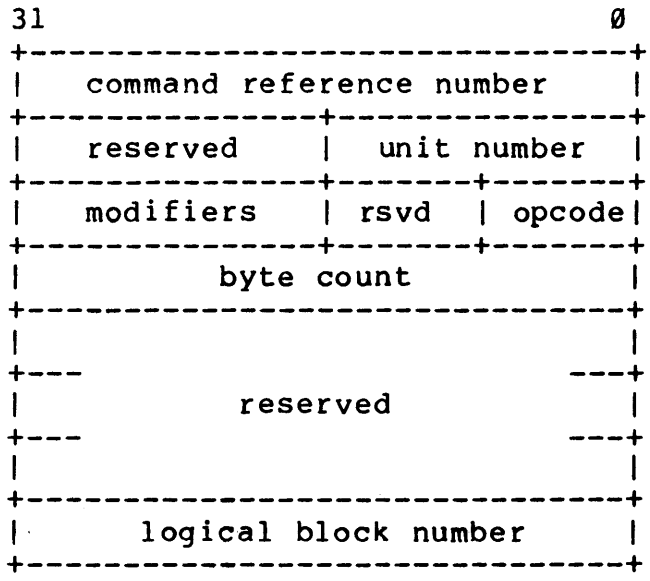
A controller may, at its option, treat this as either an Immediate, Sequential, or Non-sequential command. Host command timeout algorithms must treat this as a non-Immediate command.

6.9 ERASE Command

Command category:

Non-sequential

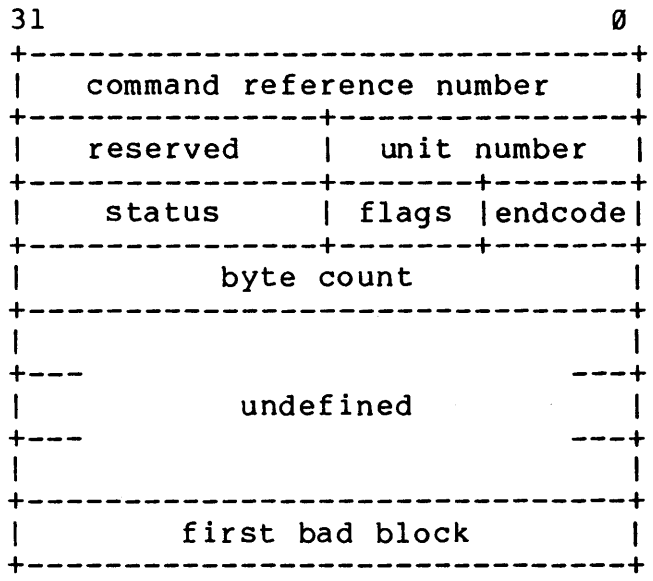
Command message format:



Allowable modifiers:

Express Request
Force Error
Suppress Error Recovery

End message format:



Status Codes:

Success (sub-code "Normal")
Success (sub-code "Duplicate Unit Number")
Invalid Command (sub-code "Invalid Byte Count")
Invalid Command (sub-code "Invalid Logical Block Number")
Command Aborted
Unit-Offline
Unit-Available
Write Protected
Controller Error
Drive Error

Description:

All data in the specified region of the unit is erased by overwriting it with zero.

This command is exactly equivalent in function, although not in performance, to a WRITE command which references a buffer that the host has zeroed.

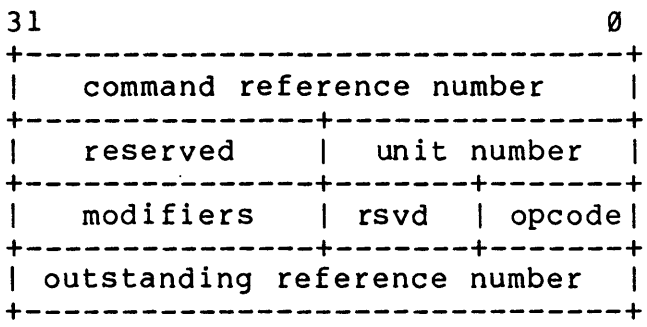
6.10 This section deliberately omitted

6.11 GET COMMAND STATUS Command

Command category:

Immediate

Command message format:



unit number

Must be the same as the "unit number" field in the outstanding command whose status is to be obtained. This allows controllers to optimize their search for the outstanding command. If the incorrect unit number is supplied, some controllers may erroneously conclude that the command is no longer outstanding, leading to erroneous command timeouts.

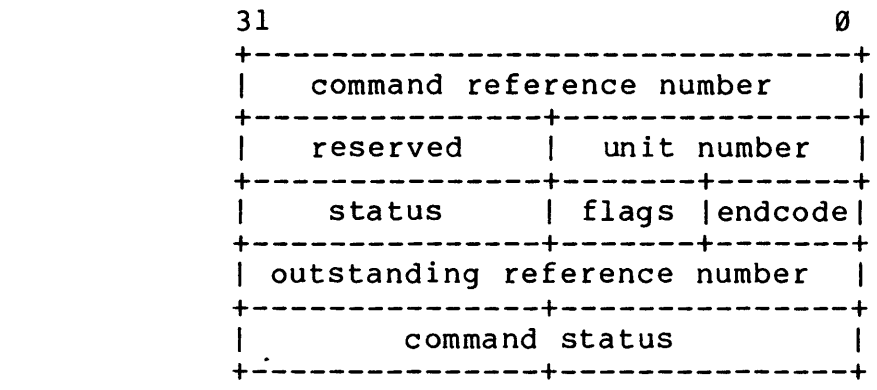
outstanding reference number

The command reference number of the command whose status is to be obtained. Note that a class driver may only obtain the status of commands that it itself has issued. This derives from the fact that command reference numbers are implicitly qualified by the connection on which the command was issued, so that class drivers have no way of naming commands issued by a different class driver.

Allowable modifiers:

none

End message format:



outstanding reference number

The command reference number of the command whose status has been returned. Identical to the value supplied in the GET COMMAND STATUS command message.

command status

The amount of work remaining to be done to complete the command, expressed as an unsigned integer. This field is zero if the command is not known to the controller, such as when the command has already completed. This field should also be zero if the command has been aborted. The controller may also return zero in this field if it can guarantee that the command will complete within the controller timeout interval. The controller must never return a value of all ones (2**32-1) in this field, as that value is used to initialize the command timeout algorithm.

The units in which this value is measured are arbitrary and may be controller, device type, and/or command dependent. However, the units must remain the same for a particular command for as long as that command is outstanding.

Status Codes:

Success (sub-code "Normal")

Description:

The GET COMMAND STATUS command is used to monitor the progress of a command towards completion. The command status measures the "doneness" of the command; the command status field is guaranteed to not increase over time. Furthermore, the command status of an MSCP server's oldest outstanding

command is guaranteed to decrease within the controller timeout interval. This last feature may be used by a host class driver to detect an insane or malfunctioning controller; see Section "Command Timeouts" for more details.

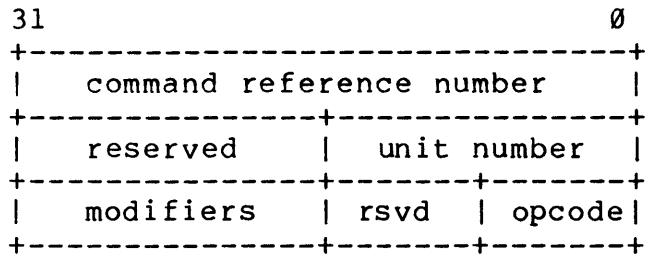
The GET COMMAND STATUS command always succeeds. If the command referenced by the "outstanding reference number" is not known to the MSCP server or has been aborted, then the MSCP server should return zero for its command status. The MSCP server may also return zero as the command status of any command that will always complete within the controller timeout interval. The MSCP server always returns the "Normal" status code in the GET COMMAND STATUS command's end message.

6.12 GET UNIT STATUS Command

Command category:

Immediate

Command message format:



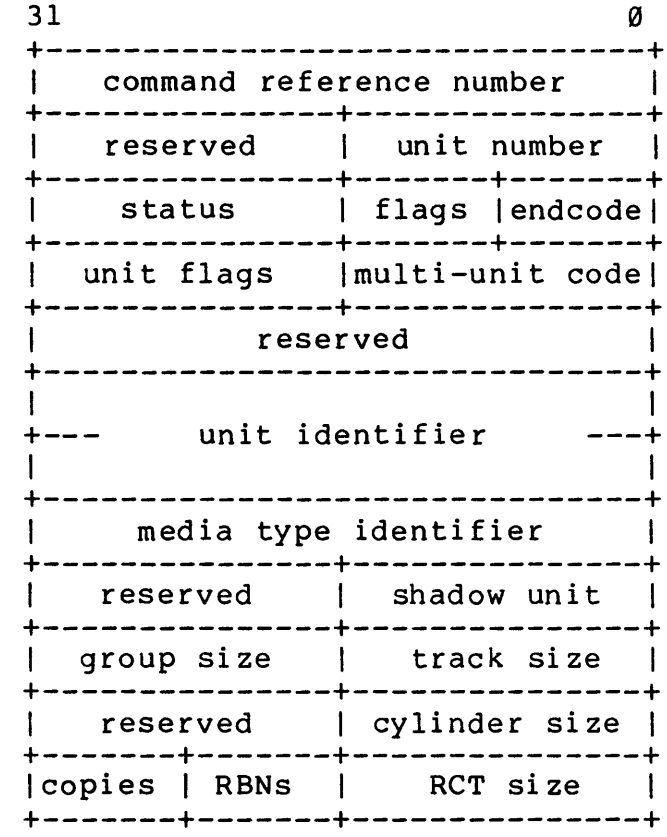
Allowable modifiers:

Next Unit

Requests that the controller return the status of the next unit (in order of ascending unit numbers) that the controller knows to exist and whose unit number is greater than or equal to the unit number specified in the command. See Section "Unit States", for a detailed definition of which units must be acknowledged by this modifier.

If this modifier is specified, the "unit number" field in the end message corresponds to the unit whose characteristics are being returned, and is typically not the same as the "unit number" field in the command message. If there are no units that are both known to the controller and whose unit numbers are greater than or equal to the unit number specified in the command message, then zero is returned in the "unit number" field of the end message. The remaining fields of the end message are identical to the values that would be returned for a GET UNIT STATUS command with a "unit number" of zero and the "Next Unit" modifier left unspecified.

End message format:



The validity of the unit characteristics returned by this command varies with the unit's state. Class drivers can determine which characteristics are valid by examining the values returned in the "status" and "unit identifier" fields. See the description below.

status

Encodes the current unit state ("Unit-Offline", "Unit-Available", "Unit-Online"). See status codes listed below.

multi-unit code

The low byte of this field identifies the access path between the controller and the unit. The high byte of this field identifies the spindle, within the access path, to which the unit belongs. See Section "Multi-unit Drives and Formatters" for more information.

unit flags

See Section "Unit Flags" under "MSCP Control Message Formats"

unit identifier

Uniquely identifies the unit among all devices accessible via MSCP; see Section "Controller and Unit Identifiers".

media type identifier

Identifies the type of media used by this unit, for use by host generic device allocation mechanisms; see Section "Media Type Identifiers".

shadow unit

Always identical to the "unit number" field.

track size

The number of logical blocks in a track, or zero if the concept of a track is inappropriate to this unit; see Section "Disk Geometry and Format".

group size

The number of tracks in a group, or zero if the concept of a group is inappropriate to this unit; see Section "Disk Geometry and Format". Note that this value must always be zero whenever "track size" is zero.

cylinder size

The number of groups in a cylinder, or zero if the concept of a cylinder is inappropriate to this unit; see Section "Disk Geometry and Format". Note that this value must alway be zero whenever "group size" is zero.

RCT size

The difference between the starting logical block numbers of successive copies of the unit's Replacement and Caching Table (RCT). Excepting only the last copy, this value is also the size of each copy of the RCT. See DEC Standard Disk Format.

RBNs

The number of replacement blocks per track. Note that this is the total number of replacement blocks on the unit if "track size" is zero, since then the entire unit is a single track. See DEC Standard Disk Format.

copies

The number of copies of the Replacement and Caching Table that are stored on the unit. See DEC Standard Disk

Format.

Status Codes:

Success (sub-code "Normal")
Success (sub-code "Duplicate Unit Number")

Both of these codes imply that the unit is "Unit-Online".

Unit-Offline
Unit-Available

Controller Error
Drive Error

For both of these codes the class driver should assume that the unit is "Unit-Offline".

Description:

The GET UNIT STATUS command returns the current state of a unit plus certain unit characteristics. In particular, the GET UNIT STATUS command is used to obtain host settable characteristics and those fixed unit characteristics that are not normally needed by the class driver.

Class drivers can determine which of the returned unit characteristics are valid by examining the returned "status" and "unit identifier" fields. The following cases exist:

1. "status" is "Success", implying that the unit is "Unit-Online". All characteristics are valid.

2. "status" is "Unit-Available" or "Unit-Offline" and "unit identifier" is not zero. All unit flags except for the "Removable media" flag are undefined. All other characteristics are valid.

3. "unit identifier" is zero. Only the "shadow unit" characteristic is valid. All other characteristics are undefined.

The three cases listed above are the only cases that can occur.

Rather than testing the entire quadword unit identifier, it is sufficient to merely test the high order word of the unit identifier, containing the class and model code bytes, to see if it is zero or not.

Controllers must supply valid values for all characteristics whenever the unit is "Unit-Online". Controllers must supply a non-zero unit identifier and valid values for all characteristics except those noted above whenever the unit is "Unit-Available" or the unit is "Unit-Offline" solely due to being disabled or known. Controllers may or may not, at the controller's option, provide valid characteristics when the unit is "Unit-Offline" for any other reason.

The rules in the above paragraphs can be restated as follows:

1. If "status" is "Success", then "unit identifier" must be non-zero and all characteristics must be valid.

2. If "status" is "Unit-Available", then "unit identifier" must be non-zero and almost all characteristics must be valid.

3. If "status" is "Unit-Offline" and the sole causes of the unit being offline are it being disabled or known, then "unit identifier" must be non-zero and almost all characteristics must be valid.

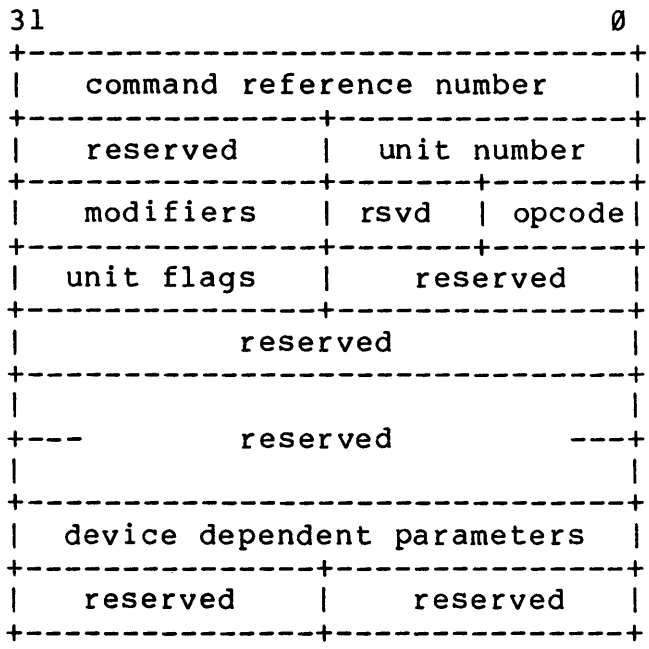
4. If "unit identifier" is zero, then "status" must either be "Unit-Offline" with some reason other than the the unit being disabled or known indicated, or "status" must be "Controller Error" or "Drive Error". Virtually no characteristics need be valid.

6.13 ONLINE Command

Command categories:

Sequential

Command message format:



unit flags

Host settable unit flags; see Section "Unit Flags" under "MSCP Control Message Formats"

If the unit is already "Unit-Online" to the class driver, then the class driver must specify the same values for controller supported host settable unit flags as are currently in effect on the unit. The controller may or may not, at its option, check that the flag values are the same. If it does check, then it must return an "Invalid Command" status code with "Invalid Unit Flags" sub-code if they are different. If the controller does not check that they are the same, then it must ignore the unit flags specified by the class driver and preserve the flag settings currently in effect on the unit. Note that this checking becomes mandatory, rather than optional, if the controller provides Multi-host Support.

device dependent parameters

Device and/or controller dependent device tuning

parameters. The value zero in this field means that default or normal tuning parameters should be used. Non-zero values for this field should normally be established through the system startup command file. Examples of the use of this field include selecting alternative optimization algorithms or enabling and disabling automatic (online) diagnosis of the unit.

Allowable modifiers:

Allow Self Destruction

Some controllers and/or drives are able to predict that a unit is in danger of imminent self destruction, and automatically spin-down and disable the unit to prevent its destruction. Such mechanisms typically sense an exponentially increasing (correctable) error rate, indicating that the disk surface has been contaminated with dust or other foreign objects. Units that have been disabled for this reason appear to be "Unit-Offline", with a sub-code indicating that they have been disabled by field service or a diagnostic. Therefore such a unit cannot normally be brought "Unit-Online".

This modifier allows a host to bring a unit that has been so disabled "Unit-Online", even though the consequences for the unit may be fatal. For this reason THIS MODIFIER MUST NEVER BE USED UNLESS FIELD SERVICE EXPLICITLY DIRECTS A SITE TO DO SO. When imminent self destruction has been predicted for a unit, it is usually possible to make one "last ditch" copy of the unit before it dies completely, recovering all or most of the data on the unit. This modifier exists primarily to simplify support of such a "last ditch" copy. This modifier also provides a means, if necessary, to work around a diagnostic that is erroneously disabling a unit.

This modifier must be supported by:

1. Any controller that may disable a unit for the reason mentioned above.

2. Any controller that may potentially be connected to a unit that can disable itself.

3. Any controller that does not disable units itself, but that will respond properly if a drive is disabled by some other (more capable) controller.

This modifier must be ignored if the unit has not been disabled or if the controller does not fall into any of the above categories.

Ignore Media Format Error

Suppresses most checking for "Media Format Errors" and causes the "576 Byte Sectors" unit flag to be host settable.

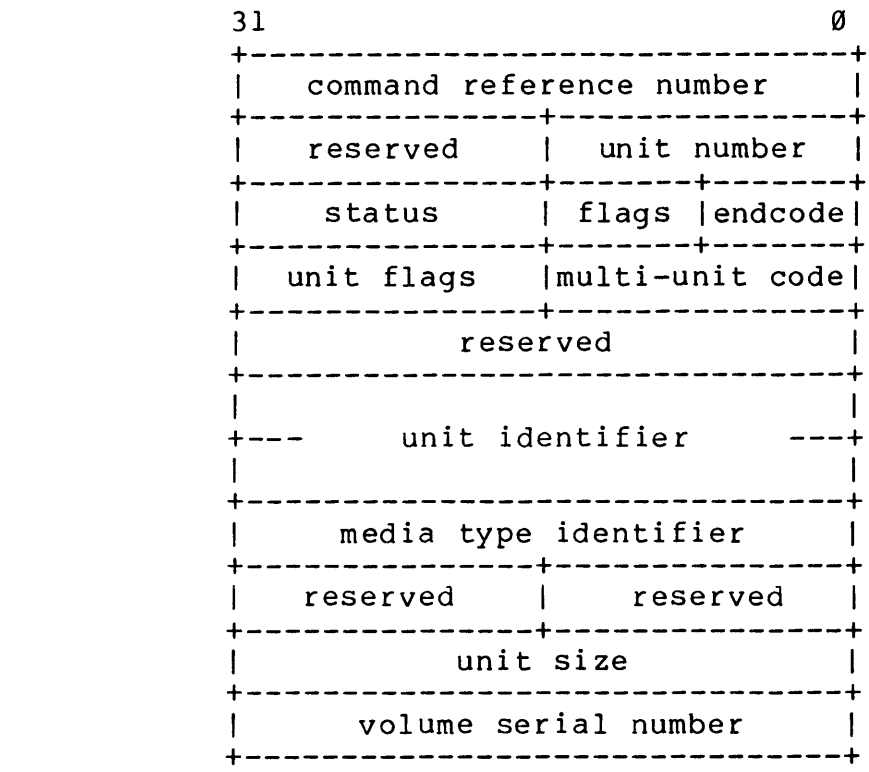
The controller uses the state of the "576 Byte Sectors" unit flag to determine whether the volume is formatted with 512 or 576 byte sectors, rather than determining the volume's format from the volume itself. The "576 Byte Sectors" unit flag will be ignored by the controller, and returned clear, if either the controller or the unit do not support 576 byte sectors.

Use of this modifier allows the host to set a unit to the wrong block or sector size. Reading a unit that is set to the wrong block or sector size may yield a mix of erroneous "Data Errors", "Drive Errors", and "Controller Errors". Writing a unit that is set to the wrong block or sector size may permanently corrupt the volume; the volume must be re-formatted if this occurs.

;Enable Set Write Protect

Causes the "Write Protect" unit flag to be host settable. This modifier causes the state of the "Write Protect (software)" unit flag to be copied to the Software Write Protect flag for this unit; see Section "Write Protection".

End message format:



The format and fields of the ONLINE command's end message are identical to the SET UNIT CHARACTERISTICS command's end message; see the field descriptions under that command. The validity of the unit characteristics returned by this command varies with the unit's state. Class drivers can determine which characteristics are valid by examining the values returned in the "status" and "unit identifier" fields.

Status Codes:

Success (sub-code "Normal")

Success (sub-code "Already Online")

The "Already Online" sub-code bit flag is set if and only if the unit is already "Unit-Online" to the requesting class driver; the unit's state and characteristics are not altered. When the unit is already "Unit-Online" to the requesting class driver, the controller merely returns the unit characteristics in the end message with this status bit flag set, without performing any other actions.

Invalid Command (sub-code "Invalid Unit Flags")

The unit is already "Unit-Online" and, for those unit flags that are both host settable and supported by the

controller, the class driver has specified different values from those currently in effect on the unit. The unit remains "Unit-Online"; the host settable unit flags are not changed. Controllers that do not provide Multi-host Support may, at their option, omit checking that the unit flags are the same. Such a controller must ignore the class driver specified unit flags for units that are already "Unit-Online", thus returning a "Success" status code with "Already Online" sub-code.

Command Aborted

The unit's state is un-changed. The host must assume that the returned unit characteristics are invalid.

Unit-Offline

Note that some causes of a unit being "Unit-Offline" may be overridden (suppressed) by the "Allow Self Destruction" command modifier.

Media Format Error

The unit is and remains "Unit-Available". However, attention messages are suppressed for this unit and the controller attempts to spin-down this unit exactly as if an AVAILABLE command with the "Spin-down" modifier set were issued. Note, however, that this error will be suppressed and the unit brought "Unit-Online" anyway if the "Ignore Media Format Error" command modifier is set. See the modifier description above.

Controller Error

The host should assume the unit is "Unit-Offline".

Drive Error

The unit is "Unit-Offline" due to being inoperative. The controller must suppress AVAILABLE attention messages for the unit and attempt to spin-down the unit, exactly as if an AVAILABLE command with the "Spin-down" modifier set were issued for the unit. The controller may subsequently report the unit as being either "Unit-Offline" or "Unit-Available"; generally the reported unit state will depend upon exactly how the unit is "broken", and the interactions of the failure with the controller's perception of the unit's state.

Description:

The ONLINE command is used to bring a unit "Unit-Online", set host settable unit characteristics, and obtain those unit characteristics that are essential for proper class driver operation. The unit is spun-up, if necessary, and its heads are loaded prior to returning the ONLINE command's end message. Host settable characteristics are set exactly as if a SET UNIT CHARACTERISTICS command were issued; see the description of that command. Host settable characteristics are set after the unit has been successfully spun-up and any other validity checks have succeeded. Note that the unit's host settable characteristics are NOT altered if the unit is already "Unit-Online".

The class driver must invoke a process that will access the unit's Replacement and Caching Table to determine if a bad block replacement operation has been partially performed or if the unit must be write protected. The details of this check and its consequences are described in Section "Bad Block Replacement" and DEC Standard Disk Format.

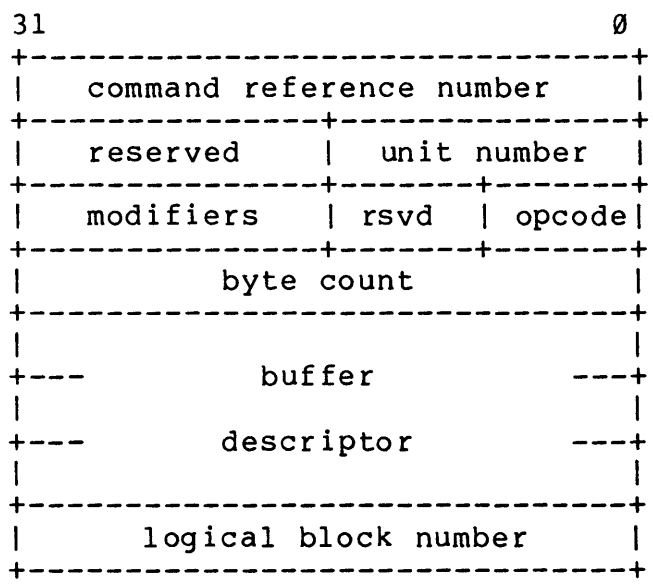
Note that the format of the ONLINE command's end message is identical to the SET UNIT CHARACTERISTICS command's end message.

6.14 READ Command

Command category:

Non-sequential

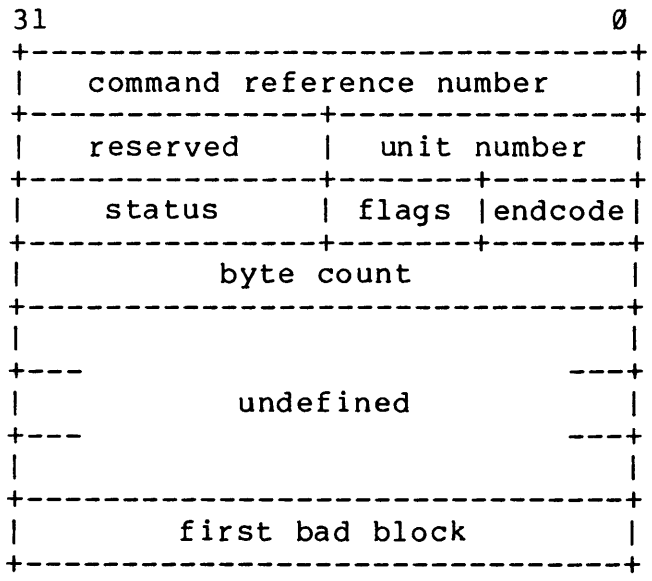
Command message format:



Allowable modifiers:

Compare
Express Request
Suppress Error Correction
Suppress Error Recovery

End message format:



Status Codes:

Success (sub-code "Normal")
Success (sub-code "Duplicate Unit Number")
Invalid Command (sub-code "Invalid Byte Count")
Invalid Command (sub-code "Invalid Logical Block Number")
Command Aborted
Unit-Offline
Unit-Available
Compare Error
Data Error
Host Buffer Access Error
Controller Error
Drive Error

Description:

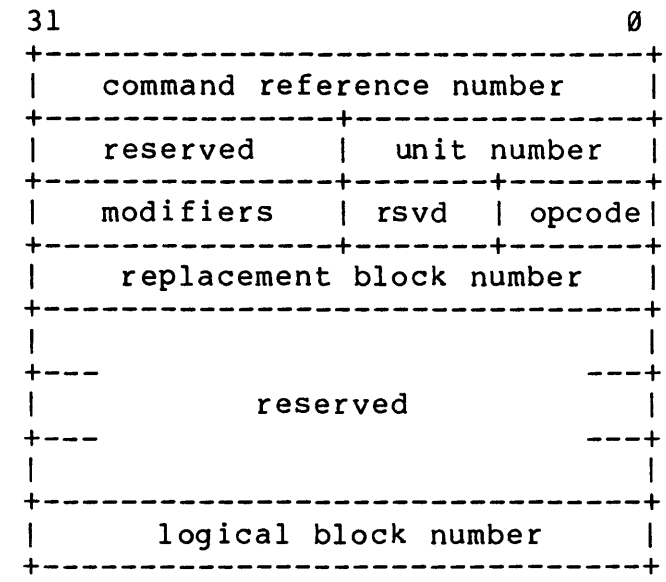
Data is read from the unit and transferred to the host buffer.

6.15 REPLACE Command

Command category:

Non-sequential

Command message format:



replacement block number

Identifies the replacement block that has been allocated to replace the bad logical block.

logical block number

Identifies the bad logical block that is being replaced.

Allowable modifiers:

Express Request

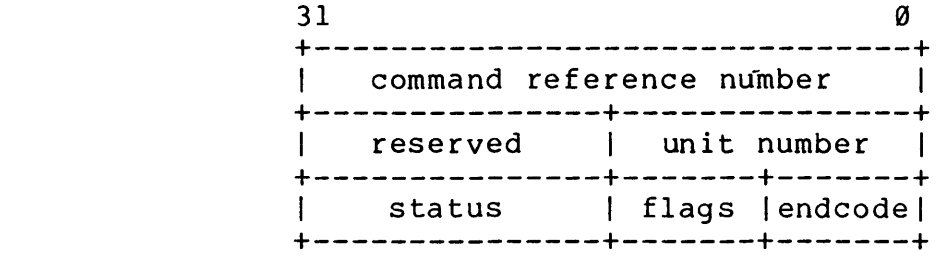
Primary Replacement Block

Must be set if and only if the "replacement block number" specifies the primary replacement block for "logical block number". That is, must be set if and only if the following expression is true:

replacement block number =
logical block number / track size * RBNs

where "track size" and "RBNs" are unit characteristics obtained via the GET UNIT CHARACTERISTICS command and "/" denotes integer (truncating) division. See DEC Standard Disk Format for more information. Note that this modifier is redundant information provided for the convenience of the controller.

End message format:



Status Codes:

Success (sub-code "Normal")
Success (sub-code "Duplicate Unit Number")
Invalid Command (sub-code "Invalid Replacement Block Number")
Invalid Command (sub-code "Invalid Logical Block Number")
Command Aborted
Unit-Offline
Unit-Available
Write Protected
Controller Error
Drive Error

Description:

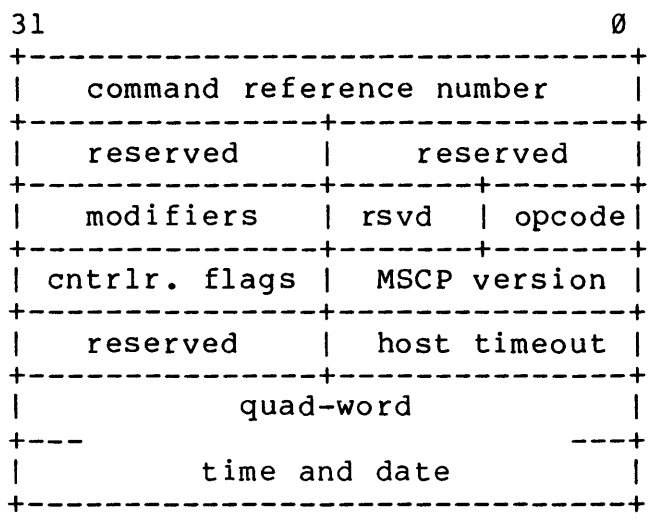
The specified logical block is flagged to indicate that it has been replaced with the specified replacement block. The volume's Replacement and Caching Table must have been updated prior to using this command, and the replacement block should be initialized with a write command to the same logical block number after using this command. See Section "Bad Block Replacement" and DEC Standard Disk Format for more information on the use and function of this command.

6.16 SET CONTROLLER CHARACTERISTICS Command

Command category:

Immediate

Command message format:



MSCP version

Host class drivers must supply the value zero in this field. MSCP servers must verify this value and, if it is not zero, return an Invalid Command end message. This value will be incremented if MSCP is ever modified in a way that is not upwards compatible.

cntrlr. flags

Host settable controller flags; see Section "Controller Flags" under "MSCP Control Message Formats"

host timeout

The time interval that the controller should use for the host access timeout with this host, or zero if the controller should disable the host access timeout for this host. Expressed as an unsigned binary integer in units of seconds. Controllers should use a default host access timeout of 60 seconds if they have not received a SET CONTROLLER CHARACTERISTICS command since becoming "Controller-Online".

Even though this is a sixteen bit field, controllers may treat all values greater than 255 as if 255 had been specified, and all values between 1 and 9 as if 10 had been specified. See Section "Host Access Timeouts" for a description of host access timeouts.

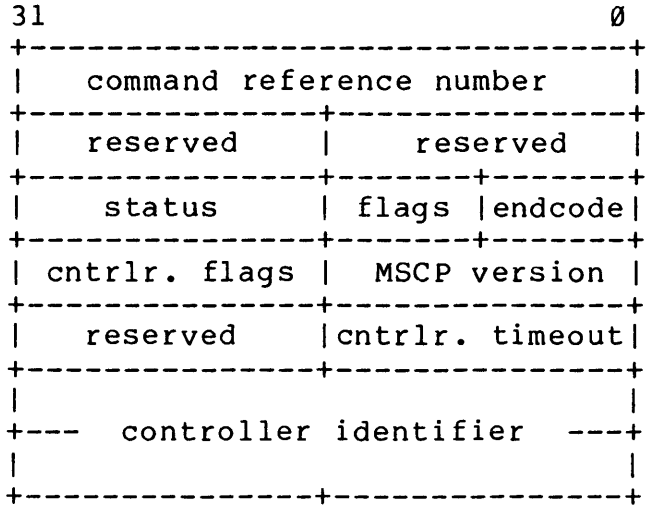
quad-word time and date

The current time and date, expressed as the number of clunks since 00:00 o'clock, November 17, 1858 (in the local time zone), or zero if the current time and date is not available. A clunk is 100 nanoseconds. This is the standard VAX/VMS time and date format. The use that is made of the current time and date and the action taken if it is not supplied (i.e., if zero is supplied) is controller dependent, and should be described in each controller's Functional Specification. Controllers must not require that a time and date be supplied for proper operation.

Allowable modifiers:

none

End message format:



cntrlr. flags

See Section "Controller Flags" under "MSCP Control Message Formats"

cntrlr. timeout

The controller timeout interval; the minimum amount of time that the controller needs to guarantee it will

accomplish useful work on its oldest outstanding command. Expressed as an unsigned binary integer in units of seconds. This value must not exceed 255 (one byte), even though a sixteen bit field has been provided. See Section "Command Timeouts".

controller identifier

Uniquely identifies the controller among all devices accessible via MSCP. See Section "Controller and Unit Identifiers".

Status Codes:

Success (sub-code "Normal")

Description:

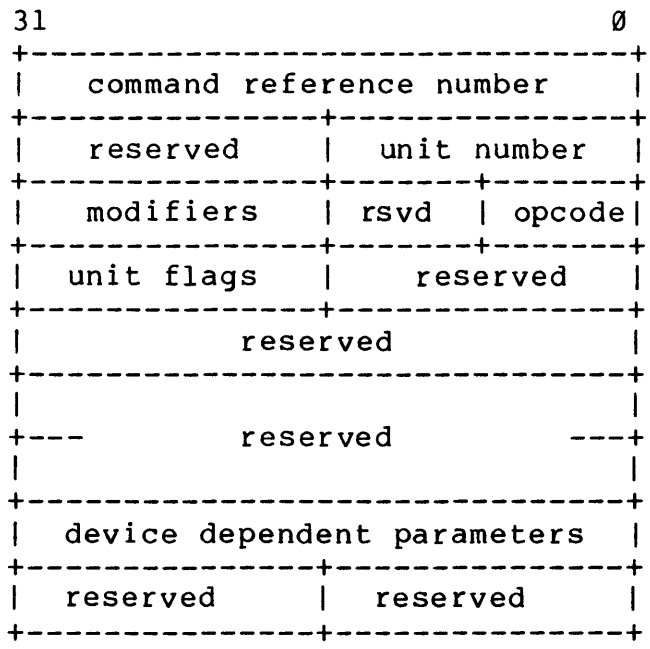
The SET CONTROLLER CHARACTERISTICS command is used to set and obtain controller characteristics. The default value for "cntrlr. flags" is all flags clear (i.e., all messages disabled); the default value for "host timeout" is 60 seconds. These default values are used from the time that the controller becomes "Controller-Online" to a host until it stops being "Controller-Online" or until the host issues a SET CONTROLLER CHARACTERISTICS command.

6.17 SET UNIT CHARACTERISTICS Command

Command category:

Sequential

Command message format:



unit flags

Host settable unit flags; see Section "Unit Flags" under "MSCP Control Message Formats"

device dependent parameters

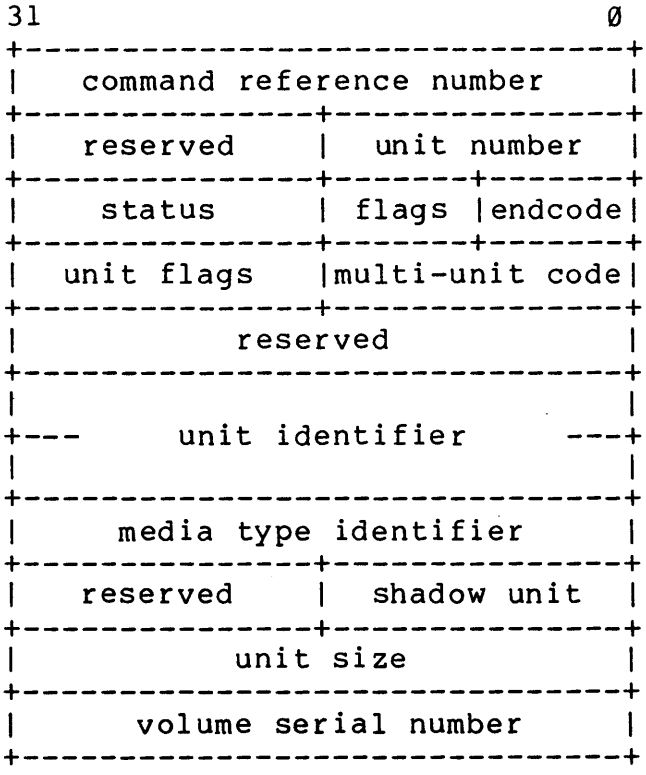
Device and/or controller dependent device tuning parameters. The value zero in this field means that default or normal tuning parameters should be used. Non-zero values for this field should normally be established through the system startup command file. Examples of the use of this field include selecting alternative optimization algorithms or enabling and disabling automatic (online) diagnosis of the unit.

Allowable modifiers:

Enable Set Write Protect

Causes the "Write Protect" unit flag to be host settable. This modifier causes the state of the "Write Protect (software)" unit flag to be copied to the Software Write Protect flag for this unit; see Section "Write Protection".

End message format:



The validity of the unit characteristics returned by this command and the ONLINE command varies with the unit's state. Class drivers can determine which characteristics are valid by examining the values returned in the "status" and "unit identifier" fields. See the description below.

multi-unit code

The low byte of this field identifies the access path between the controller and the unit. The high byte of this field identifies the spindle, within the access path, to which the unit belongs. See Section "Multi-Unit Drives and Formatters", for more information.

unit flags

See Section "Unit Flags" under "MSCP Control Message Formats"

unit identifier

Uniquely identifies the unit among all devices accessible via MSCP; see Section "Controller and Unit Identifiers".

media type identifier

Identifies the type of media used by this unit, for use by host generic device allocation mechanisms; see Section "Media Type Identifiers".

shadow unit

Always identical to the "unit number" field.

unit size

The number of logical blocks in the host area of this unit. This value does NOT include the logical block range occupied by the unit's Replacement and Caching Table. The logical block number of the first block of the unit's Replacement and Caching Table is equal to this value.

volume serial number

The low order 32 bits of the serial number of the volume that is mounted on this unit. Zero if the volume does not have a serial number. When displayed in human readable form, this number should be formatted as a ten digit decimal number with leading zeros printed. Undefined if the unit is "Unit-Offline" or "Unit-Available", or if the "Ignore Media Format Error" modifier was set in the ONLINE command that brought this unit "Unit-Online".

Hosts must not assume that the value returned in this field uniquely identifies the volume. Although this value often will be unique, the uniqueness is not guaranteed, especially across media obtained from several independent suppliers. Also, media that must meet external (industry compatible) format standards will typically be unable to implement a volume serial number; this field will always be zero for such media.

Status Codes:

Success (sub-code "Normal")
Success (sub-code "Duplicate Unit Number")

Imply that the unit is "Unit-Online".

Command Aborted

The unit's state is un-changed. The host must assume

that the returned unit characteristics are invalid.

Unit-Offline
Unit-Available

Controller Error
Drive Error

For both of these status codes the class driver should assume that the unit is "Unit-Offline".

Description:

The SET UNIT CHARACTERISTICS command is used to set host settable unit characteristics and obtain those unit characteristics that are essential for proper class driver operation. This command never alters the unit's state ("Unit-Online", "Unit-Available", "Unit-Offline"). It is meaningless to set host settable characteristics for a unit that is "Unit-Available" or "Unit-Offline".

The ONLINE command performs a SET UNIT CHARACTERISTICS operation after bringing a unit "Unit-Online".

Class drivers can determine which of the returned unit characteristics are valid by examining the returned "status" and "unit identifier" fields. The following cases exist:

1. "status" is "Success", implying that the unit is "Unit-Online". All characteristics are valid. Note that the value of "volume serial number" is undefined if the unit was brought online by an ONLINE command with the "Ignore Media Format Error" modifier.

2. "status" is "Unit-Available" or "Unit-Offline" and "unit identifier" is not zero. All unit flags except for the "Removable media" flag are undefined and the "volume serial number" is undefined. All other characteristics are valid.

3. "unit identifier" is zero. Only the "shadow unit" characteristic is valid. All other characteristics are undefined.

The three cases listed above are the only cases that can occur.

Rather than testing the entire quadword unit identifier, it is sufficient to merely test the high order word of the unit identifier, containing the class and model code bytes, to see if it is zero or not.

Controllers must supply valid values for all characteristics whenever the unit is "Unit-Online". Controllers must supply a non-zero unit identifier and valid values for all characteristics except those noted above whenever the unit is "Unit-Available" or the unit is "Unit-Offline" solely due to being disabled or known. Controllers may or may not, at the controller's option, provide valid characteristics when the unit is "Unit-Offline" for any other reason.

The rules in the above paragraphs can be restated as follows:

1. If "status" is "Success", then "unit identifier" must be non-zero and all characteristics must be valid.

2. If "status" is "Unit-Available", then "unit identifier" must be non-zero and almost all characteristics must be valid.

3. If "status" is "Unit-Offline" and the sole causes of the unit being offline are it being disabled or known, then "unit identifier" must be non-zero and almost all characteristics must be valid.

4. If "unit identifier" is zero, then "status" must either be "Unit-Offline" with some reason other than the the unit being disabled or known indicated, or "status" must be "Controller Error" or "Drive Error". Virtually no characteristics need be valid.

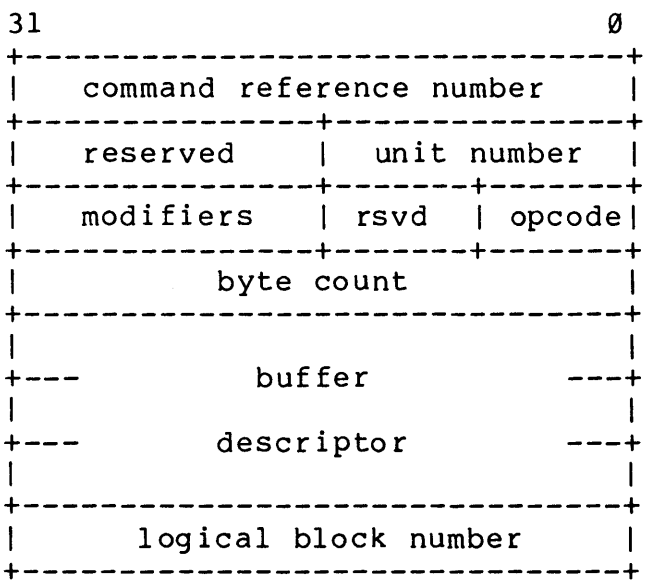
Note that the format of the SET UNIT CHARACTERISTICS command's end message is identical to that of the ONLINE command's end message.

6.18 WRITE Command

Command category:

Non-sequential

Command message format:



Allowable modifiers:

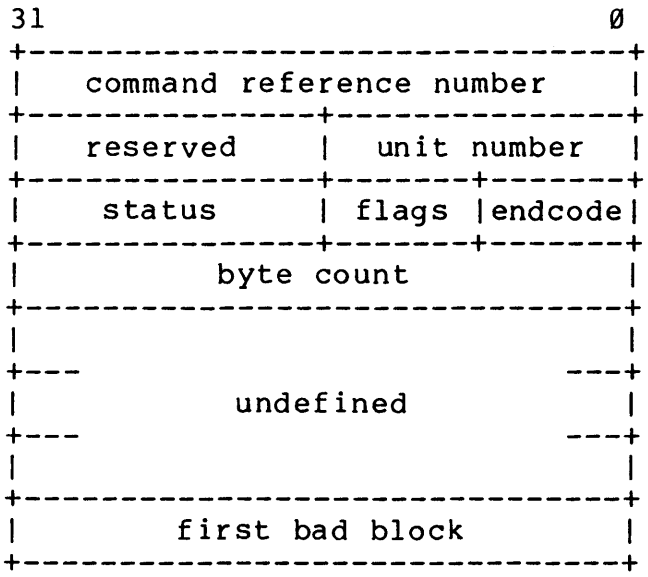
Compare
Express Request
Force Error

Suppress Error Correction

Note that this modifier only affects the compare pass of a write compare operation; it has no affect on the write operation itself.

Suppress Error Recovery

End message format:



Status Codes:

Success (sub-code "Normal")
Success (sub-code "Duplicate Unit Number")
Invalid Command (sub-code "Invalid Byte Count")
Invalid Command (sub-code "Invalid Logical Block Number")
Command Aborted
Unit-Offline
Unit-Available
Write Protected
Compare Error
Data Error
Host Buffer Access Error
Controller Error
Drive Error

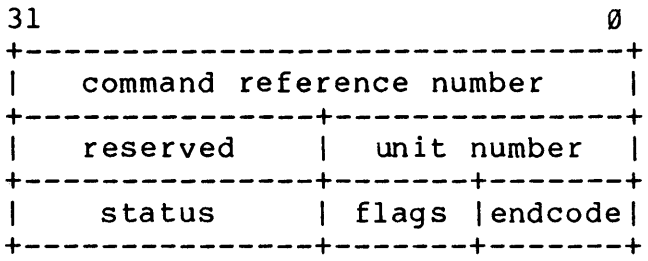
Description:

Data is fetched from the host data buffer and written to the unit.

6.19 Invalid Command End Message

The controller returns an Invalid Command end message for commands that violate the MSCP protocol.

End message format:



Status Codes:

Invalid Command (sub-code "Invalid Message Length")
Invalid Command (sub-code "Invalid MSCP Version")
Invalid Command (sub-code "Invalid Opcode")
Invalid Command (sub-code "Invalid Modifier")
Invalid Command (sub-code "Invalid Unit Flags")
Invalid Command (sub-code "Invalid Controller Flags")
Invalid Command (sub-codes used for reserved fields)

Description:

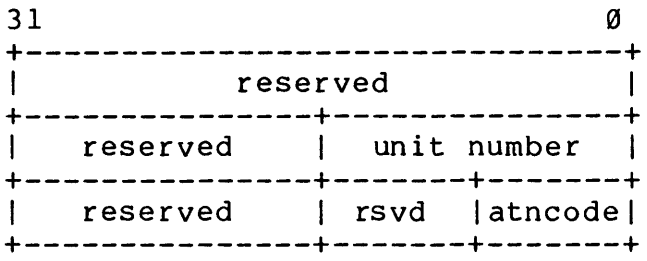
The controller returns this end message for any command that violates the MSCP protocol. Protocol violations include illegal opcodes, messages that are too short to include the parameters required by the opcode, reserved bits set in flag fields, and non-zero values in reserved fields.

The "command reference number" and "unit number" fields are copied from the illegal command message; their contents are undefined if the message was too short to contain these parameters. The "endcode" field is NOT copied from command message (since the command message does not describe a valid command). Instead, the "endcode" field always contains the constant (OP.END) defined in Table A-1.

The controller may or may not, at its option, enter the "Controller-Available" state relative to the issuing host class driver after it returns this end message.

6.20 ACCESS PATH Attention Message

Attention message format:



unit number

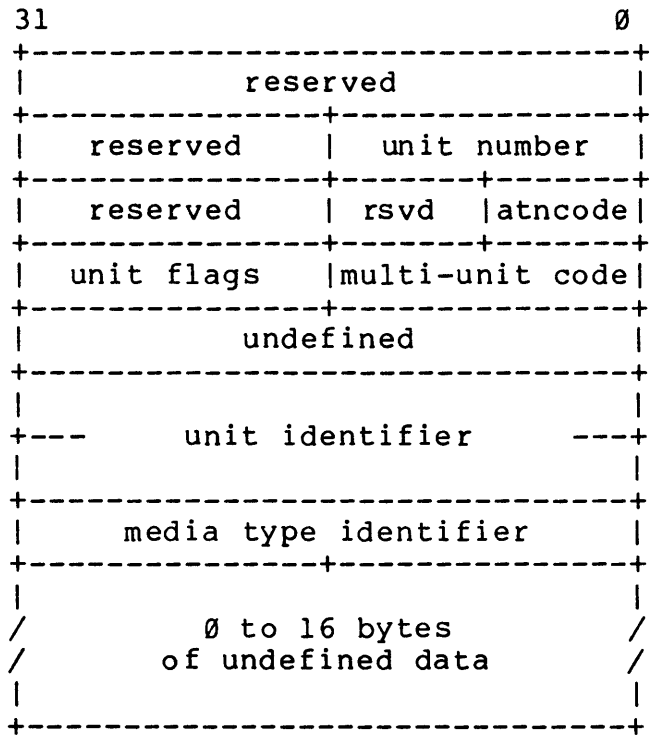
Identifies the unit for which an alternate access path is being reported.

Description:

MSCP servers use this attention message to report alternate access paths to multi-access units. This message reports that the specified unit is potentially accessible via the sending MSCP server -- i.e., it would be "Unit-Available" if it and all units that share its access path ceased being "Unit-Online" via another controller. The specific event that causes an MSCP server to send this attention message is the receipt, by the controller to which the unit is "Unit-Online", of a DETERMINE ACCESS PATHS command. This attention message is sent to all class drivers that are "Controller-Online" to the MSCP server and have enabled attention messages. See Section "Multi-Access Drives" for more information.

6.21 AVAILABLE Attention Message

Attention message format:



unit number

Identifies the unit that just became "Unit-Available".

multi-unit code
unit flags
unit identifier
media type identifer

Identical to the corresponding fields in the GET UNIT STATUS or SET UNIT CHARACTERISTICS command end messages. These fields must be valid as defined for the unit being "Unit-Available", regardless of the actual state of the unit when the message is sent. That is, the "multi-unit code", "unit identifier", and "media type identifier" must all be valid and the "Removable media" unit flag must be valid.

Description:

6.21 AVAILABLE Attention Message

An MSCP server sends an AVAILABLE attention message to a "Controller-Online" class driver when a unit asynchronously becomes "Unit-Available" to that class driver, unless AVAILABLE attention messages have been suppressed for that unit by an AVAILABLE command with the "Spin-down" modifier or by an error with similar side effects. Changes to the "Unit-Available" state due to the class driver itself issuing an AVAILABLE command are synchronous; all other changes to "Unit-Available" are asynchronous. See Section "Unit States".

The actual sending of an AVAILABLE attention message may be delayed for an arbitrarily long time, due to communications mechanism flow control, from the time that the unit actually becomes "Unit-Available". The message must not be sent if the class driver ceases to be "Controller-Online" during this delay. The message must be sent anyway if the unit, or any unit with which it shares an access path, becomes "Unit-Online" via another controller during this delay. The message may or may not be sent, at the controller's option, if the unit ceases to be "Unit-Available" for any other reason during this delay.

Note that, due to these delays, it is possible for an AVAILABLE attention message to be received after the class driver has already brought the unit "Unit-Online". Therefore class drivers must not use AVAILABLE attention messages to flag "Unit-Online" units as having become "Unit-Available". The proper procedure is to issue a command, such as a GET UNIT STATUS, to a "Unit-Online" unit for which an AVAILABLE attention message has been received, and only flag the unit as having become "Unit-Available" if the command returns that status code.

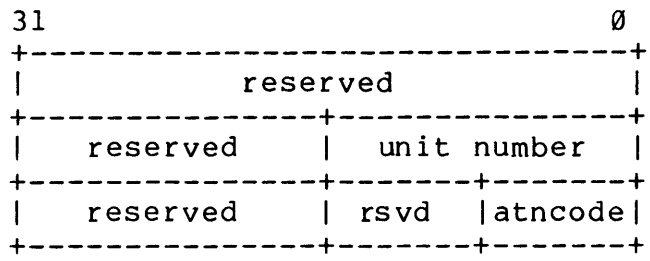
AVAILABLE attention messages are not sent for units that are already "Unit-Available" when a class driver enables attention messages. Class drivers that need to be aware of all "Unit-Available" units must enable attention messages, then scan all units via the GET UNIT STATUS command with the "Next Unit" modifier set to locate all units that are already "Unit-Available". All units that subsequently become "Unit-Available" will be reported with an AVAILABLE attention message.

An MSCP server may send redundant or erroneous AVAILABLE attention messages at any time. The frequency of such messages must be low enough that they do not represent a significant overhead for either hosts or the communications mechanism. The information contained in such messages (unit number, unit identifier, media type identifier, etc.) must correspond to an actual, physical unit that is potentially accessible via that MSCP server (i.e., connected to the controller), although the unit need not be "Unit-Available". Note that hosts must be able to handle seemingly erroneous

AVAILABLE attention messages in any case, since the unit's state may change before the host can act on an otherwise correct message.

6.22 DUPLICATE UNIT NUMBER Attention Message

Attention message format:



unit number

Identifies the unit number that is duplicated on two or more units.

Description:

An MSCP server sends DUPLICATE UNIT NUMBER attention messages to notify hosts that two or more units of the same device class have the same unit number. This allows the hosts to complain to an operator, who can correct the condition. The DUPLICATE UNIT NUMBER attention messages are sent to all hosts that are "Controller-Online" and have enabled attention messages. See Section "Unit Numbers" for a detailed discussion of the handling of duplicate unit numbers. Note that a DUPLICATE UNIT NUMBER attention message is sent regardless of whether or not one of the units is "Unit-Online".

The actual sending of a DUPLICATE UNIT NUMBER attention message may be delayed for an arbitrarily long time, due to communications mechanism flow control, from the time that the controller first detects the duplicate unit number. The message must not be sent if the class driver ceases to be "Controller-Online" during this delay. The message may or may not be sent, at the controller's option, if the duplicate unit number condition disappears during this delay.

DUPLICATE UNIT NUMBER attention messages are not sent for duplicate unit number conditions that already exist when a class driver enables attention messages. Class drivers that need to be aware of all duplicate unit number conditions must enable attention messages, then scan all units via the GET UNIT STATUS command with the "Next Unit" modifier set to locate all duplicate unit numbers. All duplicate unit numbers that the controller subsequently detects will be reported with a DUPLICATE UNIT NUMBER attention message.

6.22 DUPLICATE UNIT NUMBER Attention Message

An MSCP server may send redundant DUPLICATE UNIT NUMBER attention messages. The frequency of such messages must be low enough that they do not represent a significant overhead for either hosts or the communications mechanism. Furthermore, the duplicate unit number condition being reported must actually exist at the time the MSCP server decides to generate the DUPLICATE UNIT NUMBER attention message. Note, however, that the duplicate unit number condition may have disappeared by the time the host receives or acts upon the message.

CHAPTER 7

DISK MSCP OPTIONS

/ no options defined yet /

CHAPTER 8

MSCP ERROR LOG MESSAGE FORMATS

8.1 Introduction

MSCP controllers report errors and unusual occurences in two ways: end messages and error log messages. Unrecoverable errors are reported in the end message of the command that encountered the error. Such errors should be reported back to the program that initiated the command, so that it can take appropriate action. Additionally, all "significant" errors are reported in error log messages, so that they can be recorded in the host's error log for eventual use by Field Service. The definition of what constitutes a "significant" error is controller and/or device specific; in general, anything that would be of interest to Field Service is a "significant" error. The errors reported by error log messages may be either recoverable or unrecoverable. Note that hosts should not record errors reported in end messages in the error log; if the error is "significant", a separate error log message will be generated.

When a host receives an error log message, the host port driver passes the class driver the error log message text and the length of the error log message. In addition, the device class (e.g., disk) of the error log message is implicit in the connection on which the message was received. Note that the order of receipt of error log messages relative to end or attention messages is expressly undefined. Therefore an error log message may be received either before or after an end message that reports the same error or that has the "Error Log Generated" flag set.

MSCP servers assign a sequence number to every error log message they generate. This sequence number serves two purposes. First, if the same error log message is sent to multiple hosts, which then record it in a common error log file, it allows the multiple copies of the same message to be recognized as describing the same error. Second, if a host requests that it receive every error log message that an MSCP server generates, it allows that host to detect missing or lost error log messages by detecting gaps in the sequence numbers. This second purpose requires that the host set all three error log enable flags in the "controller flags" field of the SET CONTROLLER CHARACTERISTICS command.

Some controllers can achieve these purposes via other means, and therefore need not implement error log sequence numbers; this is further described in the third paragraph below.

Each MSCP server implements a single error log sequence number, which it uses for all error log messages for all class drivers. The server must increment its sequence number each time it attempts to generate an error log message. Multiple copies of the same error log message must all have the same sequence number. The sequence number is reset whenever the MSCP server loses context. The first error log message after such a loss of context has sequence number zero. The sequence number must not be reset as a normal result of the MSCP server becoming "Controller-Online" or "Controller-Available" to a class driver. Note that each MSCP server (i.e., each device class) within a controller has its own error log sequence number.

As stated above, the error log sequence number is only reset when the MSCP server loses context. The MSCP server reports the fact that it has lost context with a flag in the first error log message it sends to each class driver after said loss of context. This means, in effect, that the MSCP server must keep track of all class drivers to which it has sent an error log message since its last loss of context, regardless of whether or not it is currently "Controller-Online" to those class drivers.

MSCP servers that meet all of the following requirements need not implement error log sequence numbers, since their purposes are achieved via other means. The requirements are:

1. The MSCP server must never drop error log messages. That is, whenever it has an error log message to generate, it must block or deadlock until it is able to generate the message.

2. The communications mechanism between the MSCP server and the class driver must guarantee all error log messages will be delivered without loss or duplication in the order that they were generated. That is, the communications mechanism must provide the same guarantees for datagrams (error log messages) that it provides for sequential messages (control messages).

3. The MSCP server and communications mechanism must be inherently incapable of communicating with more than one class driver.

MSCP servers that meet the above three requirements may generate all error log messages with a sequence number of zero and with the "Sequence Number Reset" flag set, rather than actually implementing sequence numbers. All other MSCP servers must implement an actual error log sequence number. Such other MSCP servers may not lose context solely to reset the error log sequence number. All losses of context must be caused by some

external event such as a power failure.

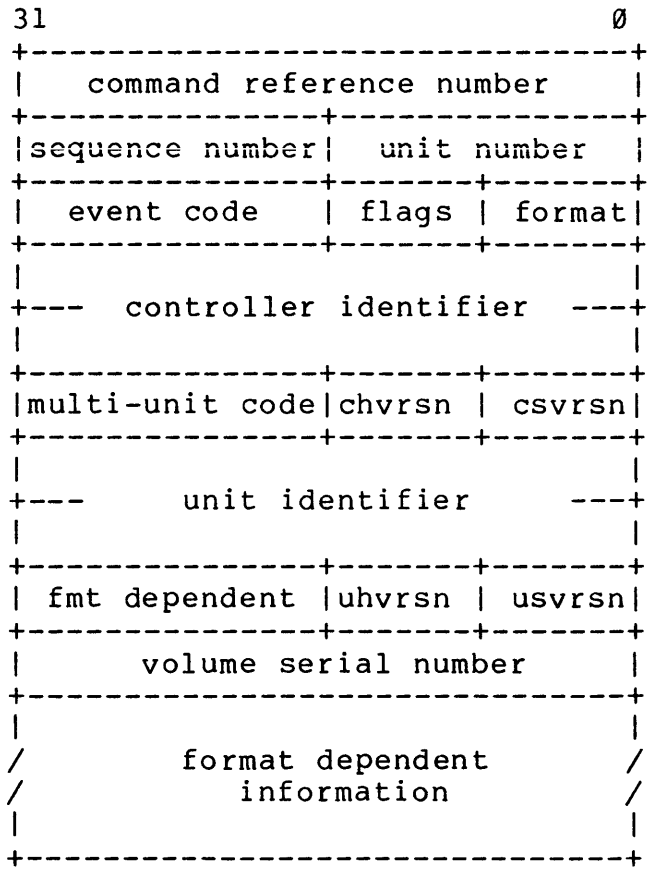
Since error log messages are not subject to flow control, it is possible for a controller to generate error log messages faster than a host can record them in its error log. In order to minimize the probability of this happening, and thus minimize the probability of losing error log information, controllers must generate no more than three error log messages in response to a single error. Note that the definition of what constitutes an error is necessarily controller and/or drive dependent. The three message limit encompasses an original error and all error recovery / correction / retry sequences associated with that error. Seemingly unrelated errors that occur in a recovery sequence are generally considered to be different errors, and are therefore not covered by the three message limit.

For example, consider an uncorrectable ECC error on a read. Re-reads using offset head positioning and the like are part of the retry sequence, and thus fall under the three message limit for the original error. However, failure of the command directing the drive to use offset head positioning (i.e., the command itself fails, indicating that the heads could not be offset) would be considered a separate error, even though both it and the original read error might have a common cause (such as a bad cable between the controller and the drive).

In order to achieve this three message limit, most error log messages summarize the results of an entire retry sequence. This approach generally results in exactly one error log message per error. The other approach is to generate a separate error log message for each attempt or retry that fails. This approach can only be used with errors for which the number of retrys to small (i.e., three or fewer attempts total), as otherwise the three message limit will be exceeded.

8.2 Generic Error Log Message Format

All MSCP error log messages must be 384 bytes or shorter in length. The actual maximum error log message size is a controller characteristic and should be described in the controller's functional specification. All host software, however, should be prepared to handle error log messages up to and including the 384 byte maximum size. The general format of error log messages is as follows:



The fields in the generic error log message are as follows:

command reference number

The command reference number of the MSCP command that caused the error reported by this error log message, or zero if the error does not correspond to a specific outstanding command. If this field contains a command reference number, then the command's end message will also have the "error log generated" end message flag set. Note that the error log message may be received either before or shortly after the command's end message.

unit number

The unit number of the unit to which the error log message relates, or zero if the message does not relate to a specific unit. This field may contain the unit number of any unit of the drive or formatter if the error relates to an entire multi-unit drive or formatter. The validity of this field is determined by the value in the "format" field.

sequence number

The sequence number of this error log message since the last time the MSCP server lost context, or zero if the MSCP server does not implement error log sequence numbers. Note that error log sequence numbers are common to all class drivers, and are not reset by class driver re-synchronization. Note also that the class driver may receive error log messages out of sequence.

format

The value in this field identifies the detailed format of the error log message, as defined in the following sections.

flags

Bit flags, collectively called error log message flags, used to report various attributes of the error. The following flags are defined:

Operation Successful

If set, the operation causing this error log message has been successfully completed. The error log message summarizes the retry sequence that was necessary to successfully complete the operation. If clear, the operation has not yet been successfully completed.

Operation Continuing

If set, the retry sequence for this operation will be continued. This error log message reports the unsuccessful completion of one or more retries. If clear, the retry sequence for this operation has terminated. Provided "Operation Successful" is also clear, the retry limit for this operation has been reached and an unrecoverable error will be reported. Undefined (meaningless) if "Operation Successful" is set.

Sequence Number Reset

If set, then the error log sequence number ("sequence number" field) has been reset by the MSCP server since the last error log message sent to the receiving class

driver. If clear, the sequence number has not been reset, implying that the "sequence number" field may be used to detect missing error log messages. Always set if the MSCP server does not implement error log sequence numbers.

If "Operation Successful" and "Operation Continuing" are both clear, then the error log message reports a hard (unrecoverable) error. If "Operation Successful" is clear and "Operation Continuing" is set, then the error log message reports an intermediate step within an error recovery operation; it is not yet certain whether the error is hard or soft. If "Operation Successful" is set, then the error log message summarizes the retry sequence used to recover from a soft error.

event code

Identifies the specific error or event being reported by this error log message. The structure of event codes is identical to the structure of the status codes returned in end messages. That is, they consist of a 5 bit major event code and an 11 bit sub-code. All errors that may be reported with both error log messages and end messages must have identical status and event codes. Also, the same value may not be used as both a status and event code unless it reports the same error as each code.

The sub-code portion of event codes is potentially controller and/or device specific. However, the same major code / sub-code combination, whenever it is used, must always have the same meaning. Therefore new sub-codes may be defined as new devices are introduced, but the meaning of old sub-codes should not change. Event code values are listed in Appendix B.

controller identifier

Uniquely identifies the controller among all devices accessible via MSCP. See Section "Controller and Unit Identifiers".

csvrsn

The controller's software, firmware, or microcode revision number.

chvrsn

The controller's hardware revision number.

multi-unit code

The multi-unit code, as defined in Section "Multi-Unit Drives

and Formatters" of the unit to which the error log message relates. This field may contain the multi-unit code of any unit of the drive or formatter if the error relates to an entire multi-unit drive or formatter.

unit identifer

Uniquely identifies the unit among all devices accessible via MSCP. See Section "Controller and Unit Identifiers". This field is only present for errors that relate to a specific unit.

usvrsn

The unit's software, firmware, or microcode revision number. This field is only present for errors that relate to a specific unit.

uhvrsn

The unit's hardware revision number. This field is only present for errors that relate to a specific unit.

volume serial number

The low order 32 bits of the serial number of the volume that is mounted on the unit. Zero if the unit's format does not provide for a volume serial number. Undefined (garbage) if there is no volume mounted in the unit, the area of the volume that contains the serial number cannot be read successfully, the error occurred before the volume serial number could be determined while bringing the unit "Unit-Online", the unit is not "Unit-Online" to any host, or if the "Ignore Media Format Error" modifier was specified in the ONLINE command that brought the unit "Unit-Online". This field is only present for errors that relate to a specific disk unit.

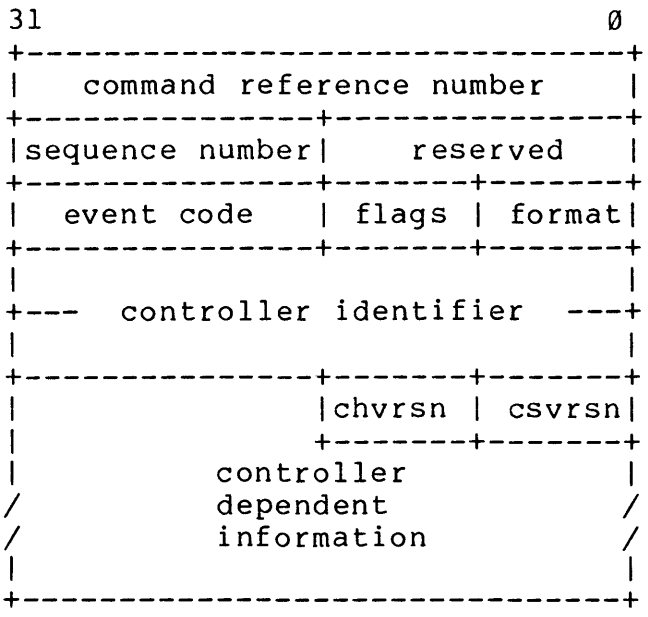
fmt dependent
format dependent information

The format of the remainder of the error log message depends upon the value of the "format" field. Note that the fields "unit number" and "multi-unit code" through "volume serial number" also depend on the value of the "format" field, as they are only present for those formats used to report errors that relate to a specific unit.

The following sections describe the specific error log message formats.

8.3 Controller Errors

The following error log message format is used to report controller errors:

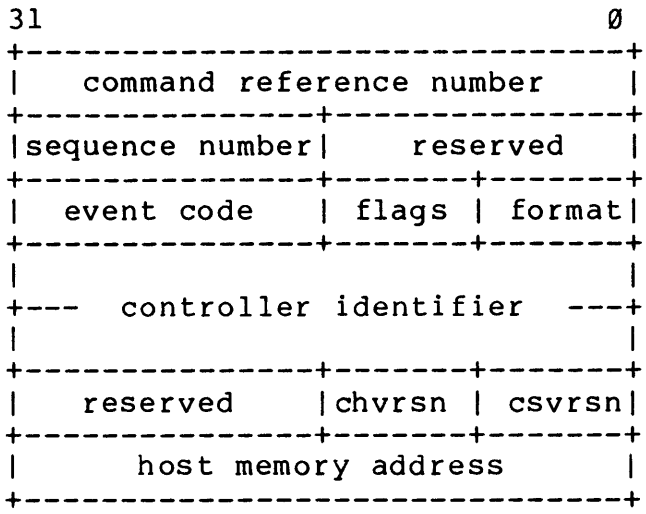


controller dependent information

A variable (controller dependent) amount of information; often no controller dependent information is provided. The length of this information is implied by the total length of the error log message, passed to the class driver by the port driver. This information will typically not be interpreted by error log formatting programs, instead being printed as a series of octal values.

8.4 Host Memory Access Errors with Bus Address

The following error log message format is used to report host memory access errors when the host memory bus address is available to the controller:

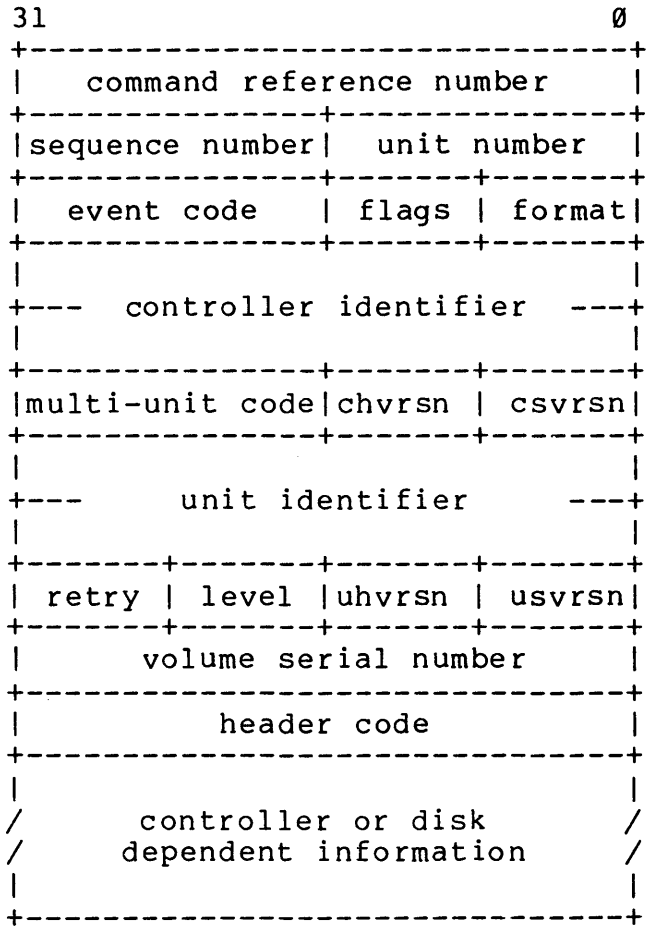


host memory address

The address on the host memory bus at which the host memory access error occurred, expressed as a 32 bit quantity. The resolution to which a controller identifies the address at which the error occurred is controller dependent, and must be described in the controller's Functional Specification. Disk controllers will typically provide a resolution of one disk block (either 512 or 576 bytes).

8.5 Disk Transfer Errors

The following error log message format is used to report errors that occur during a disk transfer. Note that this format is generally used to report the results of a sequence of retries.



level

The error recovery level used for the most recent attempt at the transfer. The error recovery level is a device dependent encoding of the special error recovery procedures, such as offset head positioning, used for the most recent transfer attempt. The values zero and 255 (all ones) are reserved to indicate that no special error recovery procedures were used.

retry

The retry count, within the current error recovery level, of the most recent attempt at the transfer. This value starts at one for the first attempt using a particular error recovery level and increments for each subsequent attempt at the same level. This continues up to some drive dependent maximum, at which time the retry count is reset to one and the next error recovery level (if any) is tried.

header code

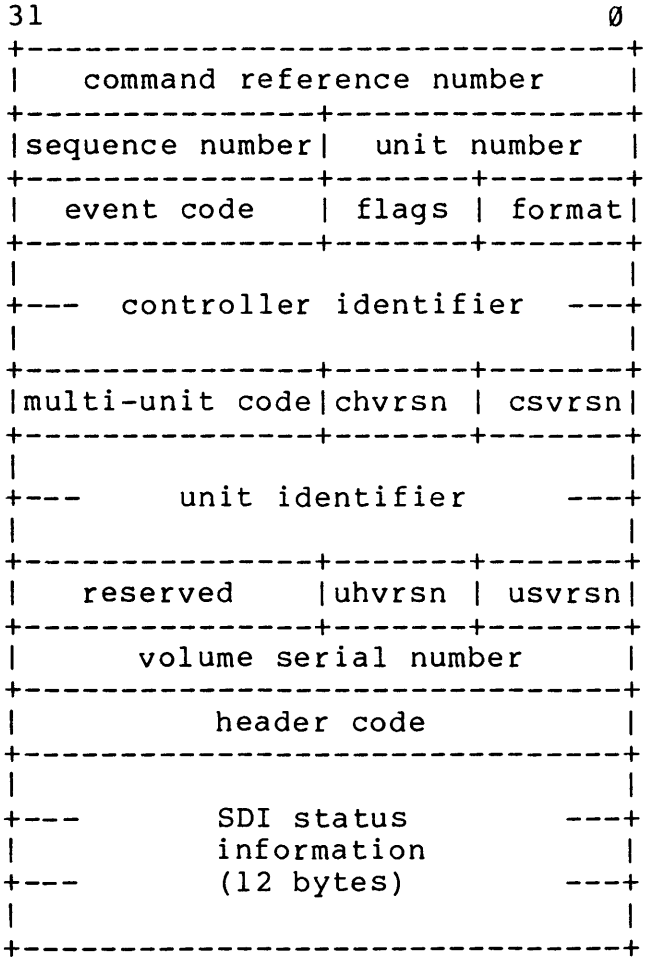
Identifies the physical disk location at which the error occurred. If the high four bits are 0000 (binary), then the low 28 bits are the logical block number at which the error occurred. If the high four bits are 0110 (binary), then the low 28 bits are the replacement block number at which the error occurred. All other patterns of the high four bits of "header code" are reserved, and must not be returned without an ECO to this specification to define their interpretation. The 28 bit logical or replacement block number may be decomposed, using the disk geometry parameters returned by the GET UNIT STATUS command, to obtain the cylinder, group, track, and sector position at which the error occurred.

controller or disk dependent information

A variable (controller or disk dependent) amount of information; often no controller or disk dependent information is provided. The length of this information is implied by the total length of the error log message, passed to the class driver by the port driver. This information will typically not be interpreted by error log formatting programs, instead being printed as a series of octal values.

8.6 SDI Errors

The following error log message format is used by SDI disk controllers to report drive detected errors and SDI communication errors. Note that the controller retries these errors only once or twice, so a separate error log message will be generated for each attempt that fails.



header code

Identifies the physical disk location at which the error occurred. If the high four bits are ØØØØ (binary), then the low 28 bits are the logical block number at which the error occurred. If the high four bits are Ø11Ø (binary), then the low 28 bits are the replacement block number at which the error occurred. All other patterns of the high four bits of "header code" are reserved, and must not be returned without an ECO to this specification to define their interpretation. The 28 bit logical or replacement block number may be decomposed, using the disk geometry parameters returned by the GET UNIT STATUS command, to obtain the cylinder, group, track, and sector position at which the error occurred.

SDI status information

Twelve bytes of status information returned by the SDI GET STATUS command or by the SDI UNSUCCESSFUL response. The unit number information returned by the SDI command or response is not included, as that information is provided elsewhere in the error log message. Otherwise, all of the SDI status information is included. See the SDI specification for the format of this information.

APPENDIX A

OPCODE, FLAG, AND OFFSET DEFINITIONS

Notes:
1. The "x" in a 32 bit mnemonic for a bit flag will be either a "V" or an "M", respectively, depending on whether the symbol is defined as a bit number (offset) or as a mask.

2. All offset values and field sizes are expressed in bytes.

Table A-1 Control Message Opcodes

Opcode Value			Preferred Mnemonics		
Dec.	Oct.	Hex.	16 bit	32 bit	Control Message Type
1	Ø1	Ø1	OP.ABO	MSCP$K_OP_ABORT	ABORT Command
16	2Ø	1Ø	OP.ACC	MSCP$K_OP_ACCES	ACCESS Command
8	1Ø	Ø8	OP.AVL	MSCP$K_OP_AVAIL	AVAILABLE Command
32	4Ø	2Ø	OP.CMP	MSCP$K_OP_COMP	COMPARE HOST DATA Command
11	13	ØB	OP.DAP	MSCP$K_OP_DTACP	DETERMINE ACCESS PATHS Command
18	22	12	OP.ERS	MSCP$K_OP_ERASE	ERASE Command
2	Ø2	Ø2	OP.GCS	MSCP$K_OP_GTCMD	GET COMMAND STATUS Command
3	Ø3	Ø3	OP.GUS	MSCP$K_OP_GTUNT	GET UNIT STATUS Command
9	11	Ø9	OP.ONL	MSCP$K_OP_ONLIN	ONLINE Command

Table A-1 Control Message Opcodes (cont.)

Opcode Value			Preferred Mnemonics		
Dec.	Oct.	Hex.	16 bit	32 bit	Control Message Type
33	41	21	OP.RD	MSCP$K_OP_READ	READ Command
20	24	14	OP.RPL	MSCP$K_OP_REPLC	REPLACE Command
4	04	04	OP.SCC	MSCP$K_OP_STCON	SET CONTROLLER CHARACTERISTICS Command
10	12	0A	OP.SUC	MSCP$K_OP_STUNT	SET UNIT CHARACTERISTICS Command
34	42	22	OP.WR	MSCP$K_OP_WRITE	WRITE Command
128	200	80	OP.END	MSCP$K_OP_END	End message flag (see note below)
7	7	7	OP.SEX	MSCP$K_OP_SEREX	Serious Exception end msg. (see below)
64	100	40	OP.AVA	MSCP$K_OP_AVATN	AVAILABLE Attention Message
65	101	41	OP.DUP	MSCP$K_OP_DUPUN	DUPLICATE UNIT NUMBER Attention Message
66	102	42	OP.ACP	MSCP$K_OP_ACPTH	ACCESS PATH Attention Message

Note: End message opcodes (also called endcodes) are formed by adding the end message flag to the command opcode. For example, a READ command's end message contains (using 16 bit mnemonics) the value OP.RD+OP.END in its opcode field. The Invalid Command end message contains just the end message flag (i.e., OP.END) in its opcode field. The Serious Exception end message contains the sum of the end message flag plus the serious exception opcode shown above (i.e., OP.SEX+OP.END) in its opcode field.

Command opcode bits 6 and 7 indicate the type of message (command, end, or attention message. Command opcode bits 3 through 5 indicate the command category (immediate, sequential, or non-sequential) and whether or not the command includes a buffer descriptor.

Table A-2 Command Modifiers

Bit Number	Bit Mask Octal	Bit Mask Hex.	Preferred Mnemonics 16 bit	Preferred Mnemonics 32 bit (see note)	Command Modifier
Generic Command Modifiers:					
14	40000	4000	MD.CMP	MSCP$x_MD_COMP	Compare
15	100000	8000	MD.EXP	MSCP$x_MD_EXPRS	Express Request
12	10000	1000	MD.ERR	MSCP$x_MD_ERROR	Force Error
9	1000	200	MD.SEC	MSCP$x_MD_SECOR	Suppress Error Correction
8	400	100	MD.SER	MSCP$x_MD_SEREC	Suppress Error Recovery
AVAILABLE Command Modifiers:					
1	2	2	MD.ALL	MSCP$x_MD_ALLCD	All Class Drivers
0	1	1	MD.SPD	MSCP$x_MD_SPNDW	Spin-down
GET UNIT STATUS Command Modifiers:					
0	1	1	MD.NXU	MSCP$x_MD_NXUNT	Next Unit
ONLINE Command Modifiers:					
0	1	1	MD.RIP	MSCP$x_MD_RIP	Allow Self Destruction
1	2	2	MD.IMF	MSCP$x_MD_IGNMF	Ignore Media Format Error
ONLINE and SET UNIT CHARACTERISTICS Command Modifiers:					
2	4	4	MD.SWP	MSCP$x_MD_STWRP	Enable Set Write Protect
REPLACE Command Modifiers:					
0	1	1	MD.PRI	MSCP$x_MD_PRIMR	Primary Replacement Block

Table A-3 End Message Flags

Bit Number	Bit Mask Octal	Bit Mask Hex.	Preferred Mnemonics 16 bit	Preferred Mnemonics 32 bit (see note)	End Message Flag
7	200	80	EF.BBR	MSCP$x_EF_BBLKR	Bad Block Reported
6	100	40	EF.BBU	MSCP$x_EF_BBLKU	Bad Block Unreported
5	40	20	EF.LOG	MSCP$x_EF_ERLOG	Error Log Generated

Table A-4 Controller Flags

Bit Number	Bit Mask Octal	Bit Mask Hex.	Preferred Mnemonics 16 bit	Preferred Mnemonics 32 bit (see note)	Controller Flag
7	200	80	CF.ATN	MSCP$x_CF_ATTN	Enable Attention Messages
6	100	40	CF.MSC	MSCP$x_CF_MISC	Enable Miscellaneous Error Log Messages
5	40	20	CF.OTH	MSCP$x_CF_OTHER	Enable Other Host's Error Log Messages
4	20	10	CF.THS	MSCP$x_CF_THIS	Enable This Host's Error Log Messages
0	1	1	CF.576	MSCP$x_CF_576	576 Byte Sectors

Table A-5 Unit Flags

Bit Number	Bit Mask Octal	Bit Mask Hex.	Preferred Mnemonics 16 bit	Preferred Mnemonics 32 bit (see note)	Unit Flag
0	1	1	UF.CMR	MSCP$x_UF_CMPRD	Compare Reads
1	2	2	UF.CMW	MSCP$x_UF_CMPWR	Compare Writes
7	200	80	UF.RMV	MSCP$x_UF_RMVBL	Removable Media
13	20000	2000	UF.WPH	MSCP$x_UF_WRTPH	Write Protect (hardware)
12	10000	1000	UF.WPS	MSCP$x_UF_WRTPS	Write Protect (software)
2	4	4	UF.576	MSCP$x_UF_576	576 Byte Sectors

Table A-6 Command Message Offsets

Offset Value			Preferred Mnemonics		Field	
Dec.	Oct.	Hex.	16 bit	32 bit	Size	Field Description
Generic Command Message Offsets:						
0	0	0	P.CRF	MSCP$L_CMD_REF	4	Command reference number
4	4	4	P.UNIT	MSCP$W_UNIT	2	Unit number
6	6	6			2	Reserved
8	10	8	P.OPCD	MSCP$B_OPCODE	1	Opcode
9	11	9			1	Reserved
10	12	A	P.MOD	MSCP$W_MODIFIER	2	Modifiers
12	14	C	P.BCNT	MSCP$L_BYTE_CNT	4	Byte count
16	20	10	P.BUFF	MSCP$Z_BUFFER	12	Buffer descriptor
28	34	1C	P.LBN	MSCP$L_LBN	4	Logical Block Number
ABORT and GET COMMAND STATUS Command Message Offsets:						
12	14	C	P.OTRF	MSCP$L_OUT_REF	4	Outstanding reference number
ONLINE and SET UNIT CHARACTERISTICS Command Message Offsets:						
12	14	C			2	Reserved
14	16	E	P.UNFL	MSCP$W_UNT_FLGS	2	Unit flags
16	20	10			12	Reserved
28	34	1C	P.DVPM	MSCP$L_DEV_PARM	4	Device dependent parameters
REPLACE Command Message Offsets:						
12	14	C	P.RBN	MSCP$L_RBN	4	Replacement block number
SET CONTROLLER CHARACTERISTICS Command Message Offsets:						
12	14	C	P.VRSN	MSCP$W_VERSION	2	MSCP version
14	16	E	P.CNTF	MSCP$W_CNT_FLGS	2	Controller flags
16	20	10	P.HTMO	MSCP$W_HST_TMO	2	Host timeout
18	22	12			2	Reserved
20	24	14	P.TIME	MSCP$Q_TIME	8	Quad-word time and date

Table A-7 End and Attention Message Offsets

Offset Value			Preferred Mnemonics		Field	
Dec.	Oct.	Hex.	16 bit	32 bit	Size	Field Description
Generic End Message Offsets:						
0	0	0	P.CRF	MSCP$L_CMD_REF	4	Command reference number
4	4	4	P.UNIT	MSCP$W_UNIT	2	Unit number
6	6	6			2	Reserved
8	10	8	P.OPCD	MSCP$B_OPCODE	1	Opcode (also called endcode)
9	11	9	P.FLGS	MSCP$B_FLAGS	1	End message flags
10	12	A	P.STS	MSCP$W_STATUS	2	Status
12	14	C	P.BCNT	MSCP$L_BYTE_CNT	4	Byte count
16	20	10			12	Reserved
28	34	1C	P.FBBK	MSCP$L_FRST_BAD	4	First bad block
ABORT and GET COMMAND STATUS End Message Offsets:						
12	14	C	P.OTRF	MSCP$L_OUT_REF	4	Outstanding reference number
GET COMMAND STATUS End Message Offsets:						
16	20	10	P.CMST	MSCP$L_CMD_STS	4	Command status
GET UNIT STATUS End Message Offsets:						
12	14	C	P.MLUN	MSCP$W_MULT_UNT	2	Multi-unit code
14	16	E	P.UNFL	MSCP$W_UNT_FLGS	2	Unit flags
16	20	10			4	Reserved
20	24	14	P.UNTI	MSCP$Q_UNIT_ID	8	Unit identifer
28	34	1C	P.MEDI	MSCP$L_MEDIA_ID	4	Media type identifier
32	40	20	P.SHUN	MSCP$W_SHDW_UNT	2	Shadow unit
36	44	24	P.TRCK	MSCP$W_TRACK	2	Track size
38	46	26	P.GRP	MSCP$W_GROUP	2	Group size
40	50	28	P.CYL	MSCP$W_CYLINDER	2	Cylinder size
42	52	2A			2	Reserved
44	54	2C	P.RCTS	MSCP$W_RCT_SIZE	2	RCT table size
46	56	2E	P.RBNS	MSCP$W_RBNS	1	RBNs / track
47	57	2F	P.RCTC	MSCP$B_RCT_CPYS	1	RCT copies

Table A-7 End and Attention Message Offsets (cont.)

Offset Value			Preferred Mnemonics		Field	
Dec.	Oct.	Hex.	16 bit	32 bit	Size	Field Description
ONLINE and SET UNIT CHARACTERISTICS End Message and AVAILABLE Attention Message offsets:						
12	14	C	P.MLUN	MSCP$W_MULT_UNT	2	Multi-unit code
14	16	E	P.UNFL	MSCP$W_UNT_FLGS	2	Unit flags
16	20	10			4	Reserved
20	24	14	P.UNTI	MSCP$Q_UNIT_ID	8	Unit identifer
28	34	1C	P.MEDI	MSCP$L_MEDIA_ID	4	Media type identifier
36	44	24	P.UNSZ	MSCP$L_UNT_SIZE	4	Unit size
40	50	28	P.VSER	MSCP$L_VOL_SER	4	Volume serial number
SET CONTROLLER CHARACTERISTICS End Message Offsets:						
12	14	C	P.VRSN	MSCP$W_VERSION	2	MSCP version
14	16	E	P.CNTF	MSCP$W_CNT_FLGS	2	Controller flags
16	20	10	P.CTMO	MSCP$W_CNT_TMO	2	Controller timeout
18	22	12			2	Reserved
20	24	14	P.CNTI	MSCP$Q_CNT_ID	8	Controller ID

Table A-8 Error Log Message Offsets

Offset Value			Preferred Mnemonics		Field	
Dec.	Oct.	Hex.	16 bit	32 bit	Size	Field Description
Generic Error Log Message Offsets:						
0	0	0	L.CRF	MSLG$L_CMD_REF	4	Command reference number
4	4	4	L.UNIT	MSLG$W_UNIT	2	Unit number
6	6	6	L.SEQ	MSLG$W_SEQ_NUM	2	Sequence number
8	10	8	L.FMT	MSLG$B_FORMAT	1	Format
9	11	9	L.FLGS	MSLG$B_FLAGS	1	Error log message flags
10	12	A	L.EVNT	MSLG$W_EVENT	2	Event code
12	14	C	L.CNTI	MSLG$Q_CNT_ID	8	Controller ID
20	24	14	L.CSVR	MSLG$B_CNT_SVR	1	Controller software version
21	25	15	L.CHVR	MSLG$B_CNT_HVR	1	Controller hardware version
22	26	16	L.MLUN	MSLG$W_MULT_UNT	2	Multi-unit code
24	30	18	L.UNTI	MSLG$Q_UNIT_ID	8	Unit ID
32	40	20	L.USVR	MSLG$B_UNIT_SVR	1	Unit software version
33	41	21	L.UHVR	MSLG$B_UNIT_HVR	1	Unit hardware version
34	42	22			2	Format dependent
36	44	24	L.VSER	MSLG$L_VOL_SER	4	Volume serial number
Host Memory Access Errors with Bus Address Error Log Message Offsets:						
24	30	18	L.BADR	MSLG$L_BUS_ADDR	4	Bus address
Disk Transfer Errors Error Log Message Offsets:						
34	42	22	L.LVL	MSLG$B_LEVEL	1	Level
35	43	23	L.RTRY	MSLG$B_RETRY	1	Retry
36	44	24	L.VSER	MSLG$L_VOL_SER	4	Volume serial number
40	50	28	L.HDCD	MSLG$L_HDR_CODE	4	Header code

Table A-8 Error Log Message Offsets (cont.)

Offset Value			Preferred Mnemonics		Field	
Dec.	Oct.	Hex.	16 bit	32 bit	Size	Field Description
SDI Errors Error Log Message Offsets:						
40	50	28	L.HDCD	MSLG$L_HDR_CODE	4	Header code
44	54	2C	L.SDI	MSLG$Z_SDI	12	SDI information

Table A-9 Error Log Message Format Codes

Format Code			Preferred Mnemonics		
Dec.	Oct.	Hex.	16 bit	32 bit	Format Description
0	0	0	FM.CNT	MSLG$K_CNT_ERR	Controller Errors
1	1	1	FM.BAD	MSLG$K_BUS_ADDR	Host Memory Access Errors with Bus Addr.
2	2	2	FM.DSK	MSLG$K_DISK_TRN	Disk Transfer Errors
3	3	3	FM.SDI	MSLG$K_SDI	SDI Errors

Table A-10 Error Log Message Flags

Bit	Bit Mask		Preferred Mnemonics		
Number	Octal	Hex.	16 bit	32 bit (see note)	Format Description
7	200	80	LF.SUC	MSLG$x_LF_SUCC	Operation Successful
6	100	40	LF.CON	MSLG$x_LF_CONT	Operation Continuing
0	1	1	LF.SNR	MSLG$x_LF_SQNRS	Sequence Number Reset

APPENDIX B

STATUS AND EVENT CODE DEFINITIONS

Notes:

1. The combination of a status or event code with a sub-code should be expressed (assuming 16 bit symbols) as: (subcode*ST.SUB)+code

2. In the sub-code tables, an asterisk in the "EV" column indicates that the code and sub-code may be used as an event code. An asterisk in the "ST" column indicates that the code and sub-code may be used as a status code.

Table B-1 Status and Event Codes

Value			Preferred Mnemonics		
Dec.	Oct.	Hex.	16 bit	32 bit	Status or Event Code
31	37	1F	ST.MSK	MSCP$M_ST_MASK	Status / event code mask
32	40	20	ST.SUB	MSCP$K_ST_SBCOD	Sub-code multiplier
0	0	0	ST.SUC	MSCP$K_ST_SUCC	Success
1	1	1	ST.CMD	MSCP$K_ST_ICMD	Invalid Command
2	2	2	ST.ABO	MSCP$K_ST_ABRTD	Command Aborted
3	3	3	ST.OFL	MSCP$K_ST_OFFLN	Unit-Offline
4	4	4	ST.AVL	MSCP$K_ST_AVLBL	Unit-Available
5	5	5	ST.MFE	MSCP$K_ST_MFMTE	Media Format Error
6	6	6	ST.WPR	MSCP$K_ST_WRTPR	Write Protected
7	7	7	ST.CMP	MSCP$K_ST_COMP	Compare Error
8	10	8	ST.DAT	MSCP$K_ST_DATA	Data Error
9	11	9	ST.HST	MSCP$K_ST_HSTBF	Host Buffer Access Error
10	12	A	ST.CNT	MSCP$K_ST_CNTLR	Controller Error
11	13	B	ST.DRV	MSCP$K_ST_DRIVE	Drive Error
31	37	1F	ST.DIA	MSCP$K_ST_DIAG	Message from an internal diagnostic

Table B-2 Standard Status and Event Sub-code Values

Sub-code	Code + Sub-code Dec.	Oct.	Hex.	E V	S T	Status or Event Sub-Code
"Success" sub-code values:						
Ø	Ø	Ø	Ø		*	Normal
1	32	40	20		*	Spin-down Ignored
2	64	100	40		*	Still Connected
4	128	200	80		*	Duplicate Unit Number
8	256	400	100		*	Already Online
16	512	1000	200		*	Still Online
"Invalid Command" sub-code values:						
Ø	1	1	1		*	Invalid Message Length
many					*	Other "Invalid Command" sub-codes values should be referenced as follows (note that this is combined with the status code): offset*256.+code where "offset" is the command message offset symbol for the field in error and "code" is the symbol for the "Invalid Command" status code.
"Command Aborted" sub-code values:						
					*	Sub-codes are not used.
"Unit-Offline" sub-code values:						
Ø	3	3	3		*	Unit unknown or online to another controller.
1	35	43	23		*	No volume mounted or drive disabled via RUN/STOP switch (unit is in known substate of Unit-Offline)
2	67	103	43		*	Unit is inoperative
4	131	203	83		*	Duplicate unit number
8	259	403	103		*	Unit disabled by field service or internal diagnostic

Table B-2 Standard Status and Event Sub-code Values (cont.)

Sub-code	Code + Sub-code Dec.	Oct.	Hex.	EV	ST	Status or Event Sub-Code
"Unit-Available" sub-code values:						
					*	Sub-codes are not used.
"Media Format Error" sub-code values:						
many				*	*	See Table B-3
"Write Protected" sub-code values:						
256	8198	20006	2006		*	Unit is Hardware Write Protected
128	4102	10006	1006		*	Unit is Software Write Protected
"Compare Error" sub-code values:						
					*	Sub-codes are not used.
"Data Error" sub-code values:						
0	8	10	8		*	Sector was written with "Force Error" modifier
many				*	*	See Table B-3
"Host Buffer Access Error" sub-code values:						
many				*	*	See Table B-3
"Controller Error" sub-code values:						
0	10	12	A			Reserved for command timeout / retry limit exceeded.
many				*	*	See Table B-3
"Drive Error" sub-code values:						
many				*	*	See Table B-3
"Message from an internal diagnostic" sub-code values:						
many				*		See Table B-3

Table B-3 Non-Standard Status and Event Sub-code Values
(Use of these sub-codes is controller or drive type dependent)

Sub-code	Code + Sub-code Dec.	Oct.	Hex.	EV	ST	Status or Event Sub-Code
"Media Format Error" sub-code values:						
1	37	45	25		*	FCT unreadable -- EDC Error
2	69	105	45		*	FCT unreadable -- Invalid sector header
3	101	145	65		*	FCT unreadable -- Data sync timeout
5	165	245	A5		*	Disk isn't formatted with 512 byte sectors
6	197	305	C5		*	Disk isn't formatted or FCT corrupted
7	229	345	E5		*	FCT unreadable -- Uncorrectable ECC Error
"Data Error" sub-code values:						
2	72	110	48	*	*	Header compare error (valid header not found)
3	104	150	68	*	*	Data Sync not found (Data Sync timeout)
7	232	350	E8	*	*	Uncorrectable ECC Error
8	264	410	108	*		One Symbol ECC Error
9	296	450	128	*		Two Symbol ECC Error
10	328	510	148	*		Three Symbol ECC Error
11	360	550	168	*		Four Symbol ECC Error
12	392	610	188	*		Five Symbol ECC Error
13	424	650	1A8	*		Six Symbol ECC Error
14	456	710	1C8	*		Seven Symbol ECC Error
15	488	750	1E8	*		Eight Symbol ECC Error
"Host Buffer Access Error" sub-code values:						
1	41	51	29		*	Odd transfer address
2	73	111	49		*	Odd byte count
3	105	151	69		*	Non-existent memory error
4	137	211	89		*	Host memory parity error

Table B-3 Non-Standard Status and Event Sub-code Values (cont.)
(Use of these sub-codes is controller or drive type dependent)

Sub-code	Code + Sub-code Dec.	Oct.	Hex.	EV	ST	Status or Event Sub-Code
"Controller Error" sub-code values:						
1	42	52	2A	*	*	SERDES overrun error
2	74	112	4A	*	*	EDC Error
3	106	152	6A	*	*	Inconsistent internal data structure.
"Drive Error" sub-code values:						
1	43	53	2B	*	*	SDI command timed out (no response or seek incomplete)
2	75	113	4B	*	*	Controller detected transmission or protocol error
3	107	153	6B	*	*	Positioner error (mis-seek)
4	139	213	8B	*	*	Lost read/write ready during or between transfers
5	171	253	AB	*	*	Drive clock dropout
6	203	313	CB	*	*	Lost receiver ready between sectors
7	235	353	EB	*	*	Drive detected error.
8	267	413	10B	*	*	Controller detected pulse or state parity error

APPENDIX C

CONTROLLER, UNIT, AND MEDIA TYPE IDENTIFER VALUES

Notes: 1. Values for new products must be added to this appendix via an ECO to this specification.

Table C-1 Controller and Unit Identifier "Class" Byte Values

Class Byte (decimal)	Subsystem Type
Ø	reserved -- must not be assigned
1	Mass storage controllers
2	Disk class devices

Table C-2 Mass Storage Controller "Model" Byte Values

Model Byte (decimal)	Controller Type
Ø	reserved -- must not be assigned
2	UDA5Ø

Table C-3 Disk Class Devices Identifier Values

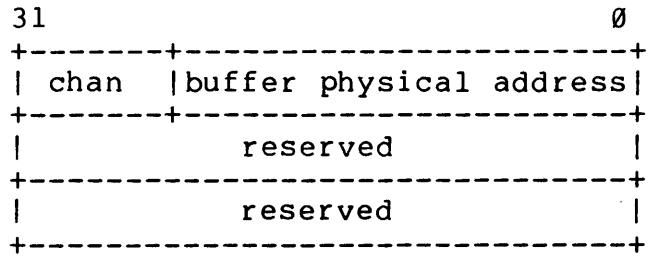
Model Byte (decimal)	Device Type Name	Media Name	Media Type Identifier octal	Media Type Identifier hex	Device
Ø			Reserved -- must not be assigned.		
1	DU	RA8Ø	Ø22544,Ø1Ø12Ø	2564,1Ø5Ø	RA8Ø fixed disk drive.

APPENDIX D

BUFFER DESCRIPTOR FORMATS

The information in this Appendix is NOT a formal part of MSCP. The formal specification for buffer descriptors is in the individual communications mechanism specifications. This information is summarized here in order to provide a convenient reference for all buffer descriptors in a single document.

The format of the buffer descriptor used with the Unibus and Q-bus is as follows:



In order to accomodate VAX-11/78Ø UBAs, it is necessary to include the UBA channel number so that controllers can request UBA purges.