Storage System Diagnostics and Utility Protocol
AA-L620A-TK
A Part of UDA50 Programmer's Doc. Kit
QP905-GZ

CHAPTER 1 INTRODUCTION

1.1 Overview of MSCP Subsystem 1-1
1.2 Overview of DUP 1-2
1.3 Purpose 1-3
1.4 Scope 1-3

CHAPTER 2 TERMINOLOGY

2.1 Terminology 2-1

CHAPTER 3 DUP HOST / CONTROLLER COMMUNICATIONS

3.1 DUP Host / Controller Communications 3-1
3.1.1 Communications Scheme 3-1
3.1.2 Class Driver / DUP Server Communications 3-1
3.1.3 Host Program / Remote Program Communcations . . 3-3

CHAPTER 4 ALGORITHIMS AND USAGE RULES

4.1 Algorithms and Usage Rules 4-1
4.1.1 DUP Server States 4-1
4.1.2 Command Categories and Execution Order 4-3
4.1.3 Class Driver / DUP Server Synchronization . . . 4-4
4.1.4 Class Driver Error Recovery 4-5
4.1.5 Command Timeouts 4-5

CHAPTER 5 GENERIC CONTROL MESSAGE FORMAT

5.1 Generic Control Message Format 5-1
5.1.1 Generic Control Message Format 5-1
5.1.2 Command Modifiers 5-3
5.1.3 End Message Format 5-4
5.1.4 Status Codes 5-4
5.1.5 ABORT PROGRAM Command / Response 5-6
5.1.6 GET DUST STATUS Command / Response 5-8
5.1.7 EXECUTE SUPPLIED PROGRAM Command / Response . 5-10
5.1.8 EXECUTE LOCAL PROGRAM Command / Response . . . 5-12
5.1.9 SEND DATA and RECEIVE DATA Commands / Responses 5-14

APPENDIX A MODIFIER CODES/RESPONSE STATUS CODES/OPCODES

APPENDIX B REMOTE PROGRAM HEADER

APPENDIX C THE DIRECT PROGRAM

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 mass storage subsystem, the controller contains the intelligence to perform the detailed I/O handling tasks. This arrangement allows the host to simply send command messages (for example, 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 communicate with the mass storage subsystem. They are the class driver and the port driver. The class driver is the higher level and is concerned with tasks being performed. The class driver's concern with details is limited to the general type of device (such as disk) and the capacity. The class driver is not concerned with the communication link (I/O bus), type of controller, or the exact model of device(s) being used.

The port driver is the lower level and is concerned only with communication services such as passing messages on and off of the communications link. The port driver is not concerned with the meaning of the messages, nor is it concerned with the exact type of controller or the exact model of storage unit(s). Thus each driver has its own level of responsibilities and shields the other from unnecessary details.

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.

1.2 Overview of DUP

While these mass storage subsystems use MSCP to perform normal data transfer operations, they use Diagnostic/Utilities Protocol (DUP) to load, start, and monitor diagnostics and utilities in the mass storage controller. This protocol provides the method of communication between the DUP class driver in the host and the DUP server in the controller for such tasks. For example, the host program uses this protocol to request that the controller load a diagnostic or utility program (either supplied by the host or from media local to the controller) and execute it. The host program may communicate with the remote program while it is running, may make inquiries to the controller about the progress of the program, and may terminate execution of the program.

The architecture is illustrated in Figure 1-1.

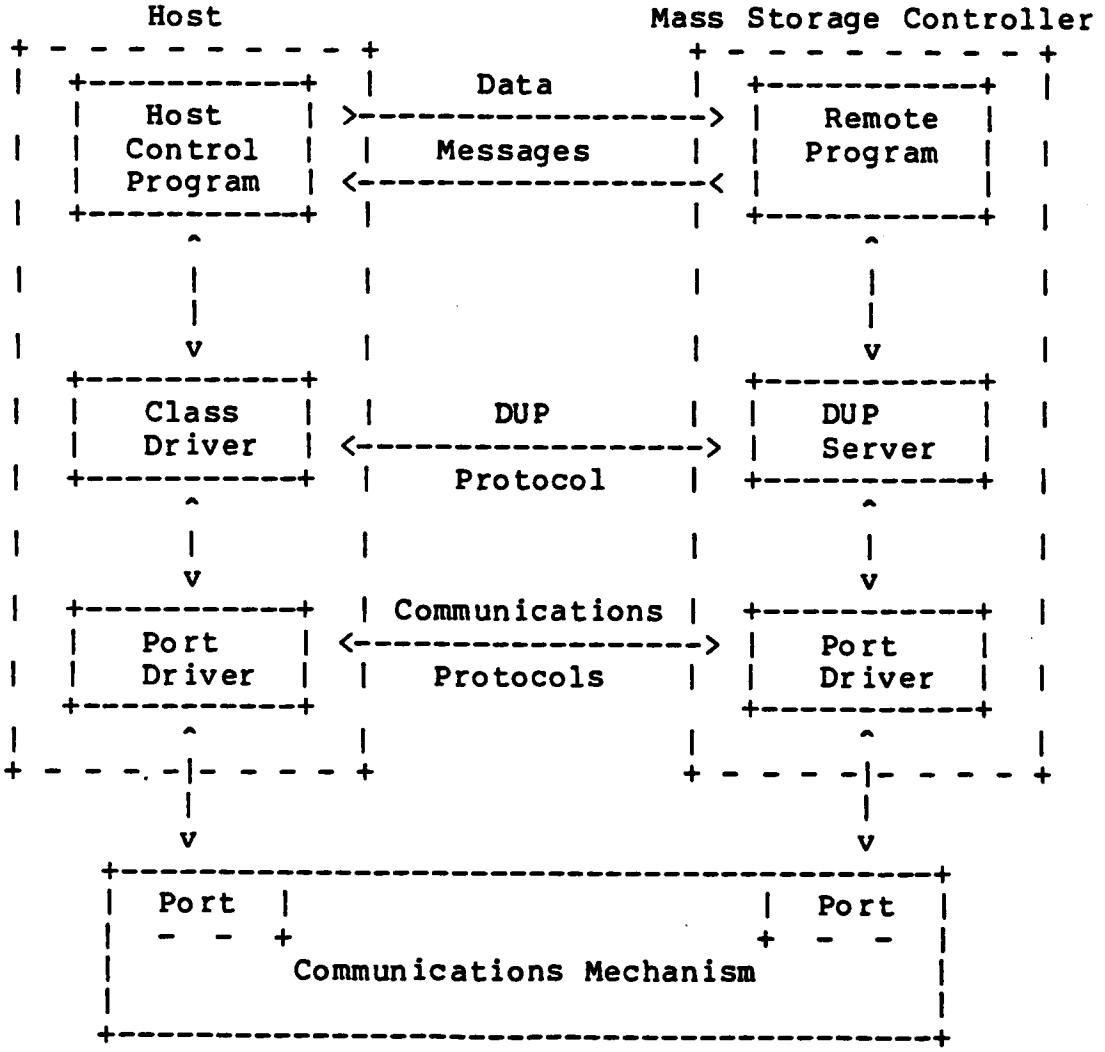


Figure 1-1 Example System

1.3 Purpose

The host class driver and the controller server communicate via a set of commands and responses. The commands and responses used by DUP are explained in this manual to the detail needed by someone writing a host DUP class driver.

1.4 Scope

The scope of this manual is limited to the details of those DUP commands and responses. This manual does not include any information on specific types of host processors or any specific operating systems. It does not assume any particular bus, controller, device type, host, or port driver implementation.

CHAPTER 2

TERMINOLOGY

2.1 Terminology

Controller Timeout Interval

The controller timeout interval is a time interval, measured in seconds, implied by the DUP command being performed or supplied by the DUP server in the GET DUST STATUS or the EXECUTE LOCAL PROGRAM command's end message.

Nugatory

Of little or no consequence: Trifling, Inconsequential.

Remote Program

The remote program is the diagnostic/utility loaded or invoked by the DUP server on behalf of the host.

Host Control Program

The host control program is the host process which loaded or invoked the remote program and provides virtual terminal services for the remote program.

Immediate Commands

Commands that DUP servers should execute immediately, without waiting for any other commands to complete.

Sequential Commands

Commands that DUP servers execute in the exact order that they were received from class drivers. Sequential commands typically allow the transfer of data between the host and the controller.

CHAPTER 3

DUP HOST / CONTROLLER COMMUNICATIONS

3.1 DUP Host / Controller Communications

3.1.1 Communications Scheme

DUP is comprised of two protocols: Class Driver/DUP Server and Host Control Program/Remote Program protocols. The Class Driver/DUP Server protocol forms the "primary" protocol; the Host Control Program/Remote Program protocol is a subprotocol which is imbeded within the SEND/RECEIVE DATA command. Both protocols use the communication mechanism described below.

3.1.2 Class Driver / DUP Server Communications

DUP runs on a dedicated connection, in the Systems Communications Architecture sense, between a host program (DUP Class Driver) and the DUP server. Flow control and connection management are provided by the underlying communications services. DUP is an asymmetric master/slave protocol, with the DUP class driver as master; i.e. DUP messages can be divided into commands and responses, where responses are only issued in response to commands and only the class driver may issue commands.

Host class drivers use the host port driver to communicate with DUP servers in controllers. DUP 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 DUP 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 connection is not established and it is believed that it cannot be established.

Two types of communications services are used across the connection between the DUP class driver and the DUP server.

- A sequential message communications service, used for DUP control messages, is used across the connection between a class driver and a DUP server. This service guarantees sequential, duplicate free delivery for all messages sent across the same connection. This service must support messages of at least 40 bytes in length.

- A block data communication service, used to move data between the host and the controller. This service provides a method of transferring the contents of a named buffer between the host and the controller or transfering data from the controller to a named buffer in the host. Buffers are identified by buffer descriptors which are provided by the host.

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 message services but have not yet been delivered.

Besides using the sequential message communications service directly, DUP uses the establishment of the connection itself to synchronize the class driver and DUP servers. The class driver will terminate the connection if it determines that it must re-synchronize with the DUP server. The DUP server may terminate the connection or it may enter the "Idle" state if re-synchronization with the class driver is necessary. Events that require 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.

Especially critical to DUP is the concept of flow control and the flow control based requirements that are imposed on DUP class drivers and DUP servers. Flow control and the flow control requirements are discussed in detail in the MSCP Specification, "Class Driver / MSCP Server Communications".

In general, 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.

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.

Note that the DUP class driver may communicate with only one DUP server per connection. However, the DUP class driver may be connected to as many DUP servers, each on a different connection, as there are DUP servers in the controller.

3.1.3 Host Program / Remote Program Communcations

The Host Program/Remote Program communications uses the block data communication services for transfer of messages and data. The details of the Host Program/Remote Program subprotocol are contained in the description of the SEND/RECEIVE DATA command.

CHAPTER 4

ALGORITHIMS AND USAGE RULES

4.1 Algorithms and Usage Rules

4.1.1 DUP Server States

The DUP server may be in any of three states relative to the DUP class driver. Each DUP server may be in a different major state relative to the class driver. The states are listed below.

Offline

The DUP server is "Offline" to the 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 server is "Offline" when it is not possible to establish a connection between the class driver and the server.

Available

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

Online

A DUP server is "Online" to the class driver whenever it can both perform operations for that class driver and the driver has synchronized with the server. A server is "Online" exactly when a connection exists between the class driver and the server; this is the state used for normal operation.

Additionally, a DUP server has two substates relative to its class driver when it is "Online".

IDLE

A DUP server is "Idle" when it is not monitoring the operation of a remote diagnostic/utility.

ACTIVE

A DUP server is "Active" when a remote diagnostic/uility is operating on behalf of the host.

The states described above actually exist between an individual DUP class driver and an individual DUP server. A host will have only one DUP driver and a subsystem may have several DUP servers. Note also that the DUP server is distinct from the state of any units connected to the controller.

A DUP server enters the "Controller-Offline" state relative to a host whenever the DUP server ceases to function or otherwise becomes unable to perform operations for the host. Possible causes are listed below.

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.

A DUP server enters the "Controller-Available" state relative to a host class driver under the following conditions.

1. The controller or DUP server is "Controller-Offline", and all causes of it being "Controller-Offline" are removed.
2. The DUP server is "Controller-Online", and the DUP server cannot successfully send a control message (i.e., a DUP end packet) to the host class driver.
3. The DUP server is "Controller-Online", and the host access timeout expires (see Section "Host Access Timeouts").
4. The host class driver terminates the connection between the class driver and the DUP server.
5. A port driver or the communications mechanism terminates the connection between the class driver and the DUP server, generally due to a communications error.

The port driver should inform the class driver whenever the DUP 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 DUP 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 a DUP 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 DUP 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.

Typically, the DUP 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 and enter the "Idle" state.

The DUP 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 DUP server by establishing a connection with the DUP server. Note that the DUP 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.

4.1.2 Command Categories and Execution Order

Most DUP commands are only legal in a particular state. The GET DUST STATUS command is the only command legal in both IDLE and ACTIVE states. Commands received while the DUP server is in an inappropriate state will be returned with the generic response status of INVALID COMMAND. The mechanism used to return the DUP server to IDLE state from the host side is the ABORT command. The mechanism used from the remote program is implementation-dependent.

A DUP server enters the ACTIVE from the IDLE state as a result of successfully performing either an EXECUTE SUPPLIED PROGRAM or EXECUTE LOCAL PROGRAM command. Returning to the IDLE state from the ACTIVE state is a result executing an ABORT PROGRAM command or as a result of remote program termination. Because the termination notification is sent only to the host control program as part of the SEND/RECEIVE DATA subprotocol, the class driver must poll the DUP server to determine if the server has returned to the IDLE state. The class driver must also determine the state of the DUP server when it receives a request from the host control program to execute a remote program as the server may have made an undetected transition to the IDLE state. Additionally, if the class driver notices that the remote program's progress indicator is not being updated, the class driver must look at the DUP

server's state indicator before deciding that the server is no longer sane.

4.1.3 Class Driver / DUP Server Synchronization

Synchronization of a class driver with a DUP server is accomplished by establishing or re-establishing the connection between the class driver and the DUP server. When the connection is established or re-established, the DUP server aborts or otherwise terminates all commands that are outstanding from that class driver. This forces the dialogue between the class driver and DUP 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 DUP server, if successful, causes the DUP 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 DUP server to a known state. DUP servers must ensure that this guarantee is met before they allow synchronization to complete (i.e., before they become "Controller-Online"). In particular, DUP servers must guarantee that no end messages will be sent and their state or context changed for any commands that were issued on an earlier incarnation of the connection between the class driver and DUP server.

Class drivers must synchronize with the DUP server whenever the host boots, recovers from a power failure, loses context, or is recovering from certain errors. After synchronizing with a DUP server, the class driver should issue a GET DUST STATUS command to establish the characteristics of the DUP server.

4.1.4 Class Driver Error Recovery

The principle method of error recovery used by class drivers is to re-synchronize with the DUP server, as described in the preceeding section. All communications mechanism failures and many controller failures are reported by terminating the connection between the class driver and DUP server, in response to which the class driver should attempt to re-synchronize with the DUP 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 DUP server. Similarly, if the DUP server decides that the class driver is insane, it enters the "Idle" state and may terminate the connection to the class driver. If the class driver is in fact actually sane, it will re-synchronize with the DUP server after the port driver notifies it that the circuit has been terminated.

4.1.5 Command Timeouts

Whenever a host issues a DUP command packet, it should time out the receipt of the corresponding response. The appropriate timeout interval for each DUP command is given in the description of the command. The DUP server may timeout receipt of host commands with a 30 second timeout and the remote program may time out the reception of SEND DATA and RECEIVE DATA commands if they desire.

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 DUP server is sane if and only if the utility or diagnostic which was initiated will ultimately complete. 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 requested diagnostic/utility. The difficulty of the host DUP class driver having to derive this "reasonable time" can be eliminated by re-stating the definition of a sane DUP server as follows: The server is sane if and only if the progress indicator is being incremented within some reasonable time. This definition allows setting "reasonable time" to some fixed value and varies the units in which we measure "useful work" according to the complexity of the diagnostic/utility. Command timeouts are based on this second definition.

A class driver implements the command timeout mechanism as follows. For each DUP server to which it is "Controller-Online", the class driver monitors the progress indicator to insure that it is changing.

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 DUP server must not be larger than 4 minutes and 15 seconds (i.e., 255 seconds).

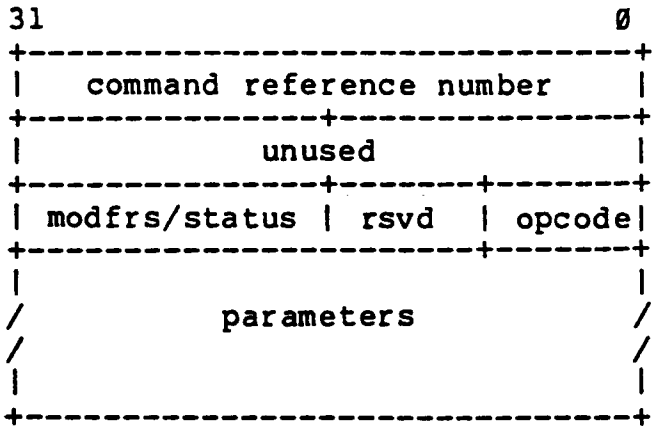
CHAPTER 5

GENERIC CONTROL MESSAGE FORMAT

5.1 Generic Control Message Format

5.1.1 Generic Control Message Format

All DUP control messages consist of a 12 byte header and a 28 byte or shorter parameter area. Multi-byte numbers are stored least significant byte first (i.e., using the standard VAX 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 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 48 bytes or that the length be an even multiple of 4 bytes. For this reason the message lengths defined by DUP are minimum lengths. Senders may pad messages as necessary to meet communications mechanism length restrictions. The contents of the padding is reserved and must follow the rules for reserved fields defined in the following paragraphs.

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 and attention messages) that they receive from an DUP server.

DUP servers must supply the value zero in the reserved fields of all end messages that they send to class drivers. DUP 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 a DUP server verifies that reserved fields are zero is controller dependent and need not be consistent for all reserved fields.

Note that the above two cases are the only allowable controller behavior for reserved fields with non-zero values. That is, if a controller may possibly respond differently depending on whether or not a reserved field is zero, then it must treat the command as invalid if the reserved field is non-zero. The only exception is reserved command message fields that correspond to end message fields in which the controller should return zeros. For such fields, the controller may merely echo the corresponding reserved fields from the command message, trusting the host to have zeroed them rather than checking and/or explicitly zeroing the fields.

A field, as used in the above discussion, may have any length. In particular, it may be an individual bit of a flag word or byte as well as an entire byte, word, or whatever. The fields in the message header are interpreted as follows shown below.

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 a DUP server. A class driver may supply a zero reference number if it does not need to associate a command with its end message.

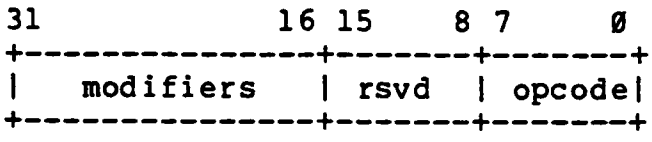
Command reference numbers must be unique across all commands that are outstanding on the same connection. That is, they must be unique across all outstanding commands issued by a single class driver (host) to a single DUP server. The class driver may re-use a command's reference number when the command is no longer outstanding -- i.e., after receiving the command's end message or after re-synchronizing with the DUP 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 DUP 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.

opcode

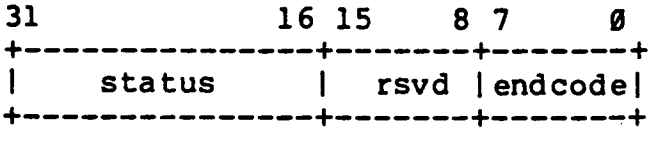
Identifies the meaning or purpose of the message. In messages sent from a class driver to a DUP server, this field specifies the operation or command to be performed. In messages sent from the controller to the class driver, this field specifies whether this is an end message or an attention message.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



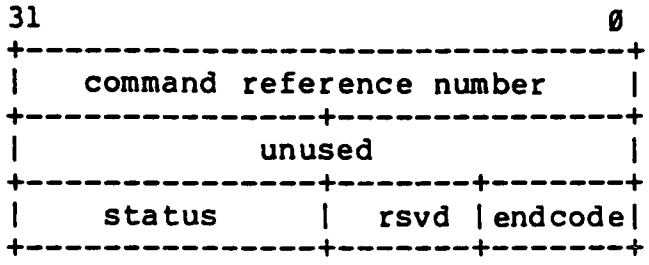
The "status" field identifies the completion status of the command. The "flags" field contains bit flags, called end flags, that report certain conditions that are disjoint from normal completion status of a command. These fields are further described in Section 5.3.

5.1.2 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 5.1. Modifiers that are only allowed on one command are described in that command's description.

5.1.3 End Message Format

A DUP 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 is copied from the command message. The remaining fields are as follows shown below.

endcode

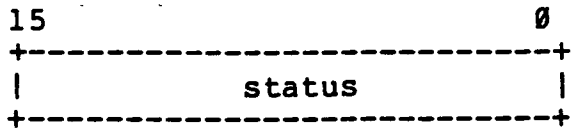
The endcode identifies this message as an end message and the type of command (opcode) that this is an end message for. This field implicitly specifies the format and interpretation of the parameters.

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.

5.1.4 Status Codes

The "status code" field is a 16-bit field as follows:



The status codes that may be returned in end message "status code" fields are listed below along with the general use made of these codes. The actual codes used are listed in Appendix B. The codes are also listed in the descriptions of the commands that may return them.

Success

The command was successfully completed.

The status code value associated with "Success" is, by definition, zero.

Invalid Command

Used to report conditions such as the state of the DUP server being incorrect for the command issued (e.g., Abort Program command issued to a server in the IDLE state) or that the command is inappropriate for the particular DUP server (eg, Execute Local Program command issued to a DUP server which does not support this feature).

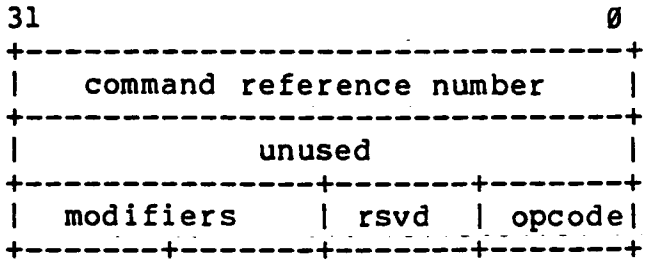
The status codes that may be returned for a specific command are command (opcode) dependent. The status codes that may be returned for each command and any special meaning that they have specific to the command are listed in the command descriptions. Note that the format of a command's end message is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

5.1.5 ABORT PROGRAM Command / Response

Command Category:

Immediate

Command message format:

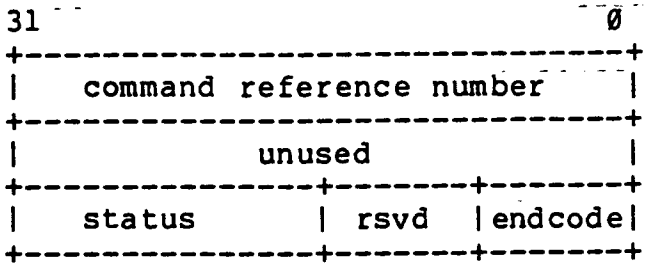


Allowable modifiers:

none

Command Opcode: 6

End message format:



Status codes:

Success
Invalid Command (server is not in ACTIVE state)

Description:

The ABORT PROGRAM command is used to terminate the execution of a remote program in an orderly fashion. When a SUCCESSFUL response is received to this command the remote program has stopped executing and the server is in IDLE state. Note that the sending of this command does not preclude further SEND DATA or RECEIVE DATA exchanges. On the contrary, the remote program may be designed to send out termination status and possibly even ask questions during its forced-exit sequence. The timeout for this command is a fixed 10 seconds, and if a response is not received

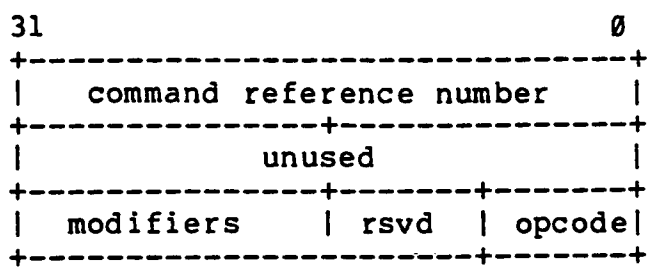
by then, the connection to the DUST should be terminated. This command is only legal if the DUST is in ACTIVE state.

5.1.6 GET DUST STATUS Command / Response

Command Category:

Immediate

Command message format:

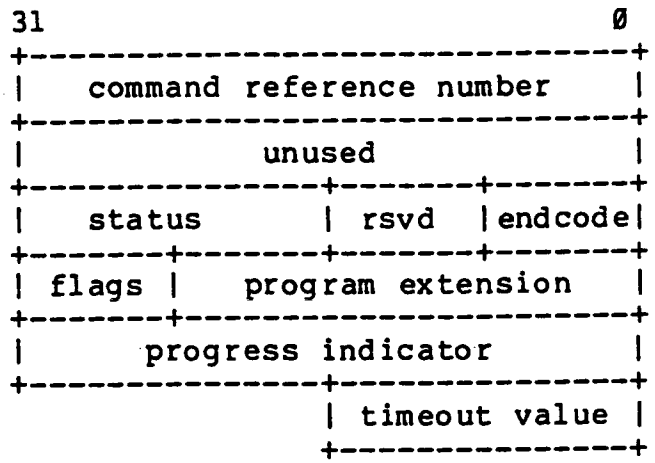


Allowable modifiers:

none

Command opcode: 1

End message format:



program extention

Extension (in ASCII) for down-line loadable programs (see appendix B).

flags

Bit 0 Set if any program execution under this server disables the operation of all other servers in the same controller.

Bit 1 Set if this controller has a local load media for loading diagnostics and utilities.

Bit 2 Set if this server will not accept the EXECUTE SUPPLIED PROGRAM Command.

Bit 3 Set if this server is currently in ACTIVE state.

progress indicator

Progress indicator for the currently running remote program; see description of the SEND DATA and RECEIVE DATA commands.

timeout

Timeout to use for the EXECUTE LOCAL PROGRAM command, in seconds; only pertinent if bit 1 of the 'flag' byte is set.

Status Codes:

Success

Description:

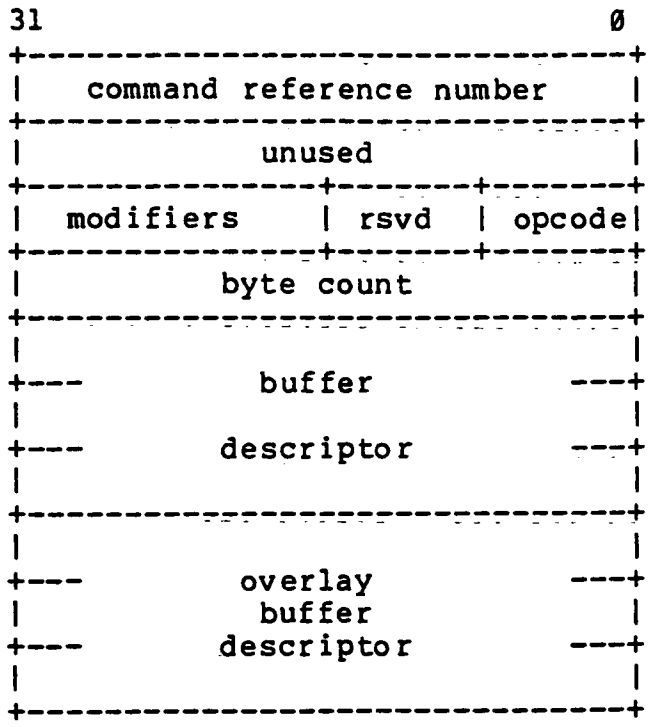
This command alows the host program to interrogate the DUP server to determine its characteristics, its state and the state of the program currently running, if any. It is legal in either IDLE or ACTIVE state and does not affect the state of server. It has a fixed timeout interval of 3 seconds. If the response times out, the host should break the connection.

5.1.7 EXECUTE SUPPLIED PROGRAM Command / Response

Command Category:

Immediate

Command message format:



byte count

Byte count for initial transfer (from bytes 0-3 of the program header)

buffer descriptor

Buffer Descriptor for initial load. This field contains the address of byte 0 of the program header.

overlay buffer descriptor

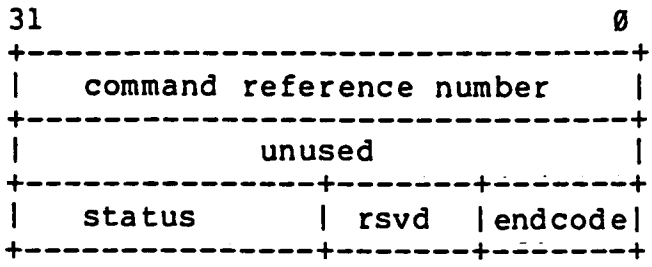
Buffer Descriptor for overlays, if required. This field contains the address of byte 0 of the overlay.

Allowable modifiers:

none

Command Opcode: 2

End message format:



Status codes:

Success
Invalid command (Server is not in IDLE state)
No region available
No region sutable
Host buffer access error (Unibus error or invalid byte count)

The format of the Buffer Descriptors shown here is dependent upon the underlying communications mechanism.

Receipt of a SUCCESSFUL response to this command means that the host may retire the buffer specified by the initial load Buffer Descriptor. The overlay buffer MUST stay assigned until execution of the remote program has terminated. Receipt of any response other than SUCCESSFUL means that both buffers may be retired.

DUP servers which do not support EXECUTE SUPPLIED PROGRAM may always return a response of INVALID COMMAND to this command.

Description:

This command causes the server to transfer the program from host memory to an area in the controller and start its execution. The host supplies the address and length (in bytes) of a buffer containing the program header (see Appendix B) and initial load. The starting address of the program, its memory requirements, and any relocation information needed to run under the server are in the program header in a format which is none of the host's business. This command is only legal when the server is in the IDLE state and return of a SUCCESSFUL end packet puts the server into the ACTIVE state.

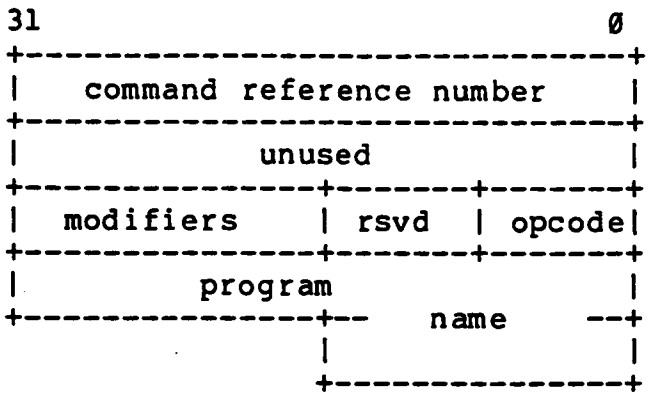
The timeout for this command is 3Ø seconds.

5.1.8 EXECUTE LOCAL PROGRAM Command / Response

Command Category:

Immediate

Command message format:



Program name

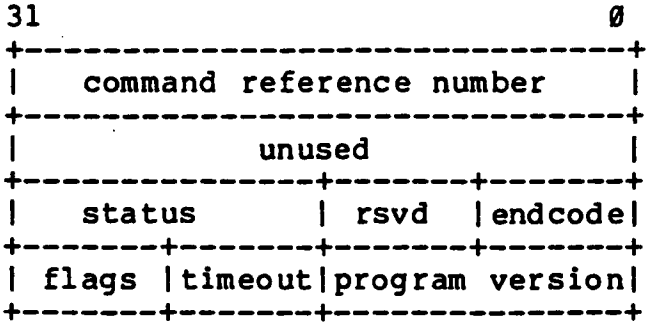
Name of local program, in ASCII, space-filled on right

Allowable modifiers are shown below.

Allow standalone - If not set, programs which have the STANDALONE characteristic will not be executed.

Command Opcode: 3

End message format:



program version

Program Version number (16 bits, binary)

timeout

Timeout value for SEND DATA and RECEIVE DATA commands in seconds. If Ø, these commands are not timed out.

flags

Flags byte of program characteristics (see appendix B)

Status codes:

Success
Invalid command (server is not in IDLE state)
No region available
No region suitable
Program not known (no such program on media)
Load failure (input error while loading program)
Standalone (STANDALONE modifier not specified for a standalone program)

** Note: servers which indicate that they do not support local programs always return a response of INVALID COMMAND to EXECUTE LOCAL PROGRAM commands.

Description:

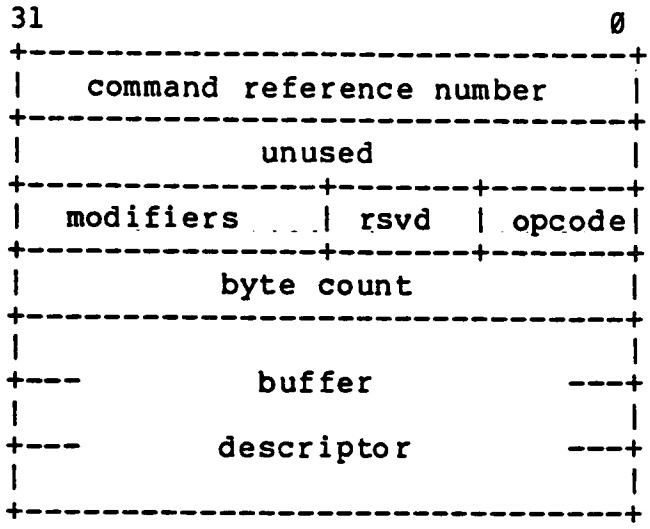
Receipt of this command causes the controller to search its local media for the named program, load it and execute it. Receipt of a SUCCESSFUL response by the host means that the program is executing and the server is in the ACTIVE state. This command is only legal when the server is in the IDLE state. The timeout value for this command is specified in the GET DUST STATUS response.

5.1.9 SEND DATA and RECEIVE DATA Commands / Responses

Command Category:

Sequential

Command message format:



byte count

byte count for transfer

buffer descriptor

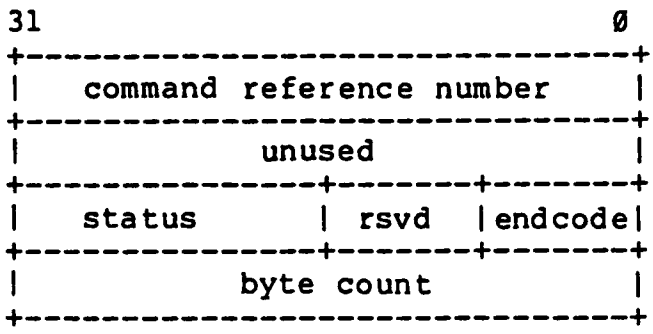
Buffer Descriptor for initial load. This field contains the address of byte 0 of the program header.

Allowable modifiers:

none

Command Opcode: 4 SEND DATA
Command Opcode: 5 RECEIVE DATA

End message format:



byte count

Number of bytes actually transfered

Status codes:

Success
Invalid Command (server is not in ACTIVE state)

Description:

These commands are used to communicate between the initiating host program and the remote program. Both commands specify a host buffer descriptor and a byte count. In the case of SEND DATA, the information in the buffer is read by the remote program and a SEND DATA response sent back to the host to acknowledge receipt. In the case of RECEIVE DATA, the remote program writes data into the buffer up to the amount specified by the byte count and then sends a RECEIVE DATA response to the host to notify it of the transmission.

The SEND DATA and RECEIVE DATA commands are only legal when the server is in the ACTIVE state. If the remote program terminates abnormally, putting the server back in the IDLE state, outstanding SEND DATA and RECEIVE DATA commands may be lost. In the event that the specified timeout interval is exceeded, the host program should issue a GET DUST STATUS command to see if the remote program is still running (ie, the DUP server is active). If it is, the Progress Indicator should be remembered and the timeout interval should be re-instated. If the second timeout expires without a response and a second GET DUST STATUS shows the remote program having made no progress in the interim (i.e. the Progress Indicator has not increased), the program should be considered broken and should be aborted.

The SEND DATA and RECEIVE DATA commands provide a general, full-duplex mechanism for communication between the host control program and a remote program under the DUP server, with the host program controlling the data flow (under the constraints of the underlying flow control mechanism). In many cases, however, it is

the remote program which wants to be "in control" of the conversation since it is where the work is being done. In addition, in any system which has at least one local terminal, the diagnostic and utility programs for that system have an existing command interface to that terminal. This generally consists of the printing of a set of prompt strings which elicit the accepting of a set of corresponding input parameters, followed by the output of the program's results and/or progress indicators. The DUP protocol provides an optional sub-protocol embedded within the SEND DATA and RECEIVE DATA messages which provides the features shown below.

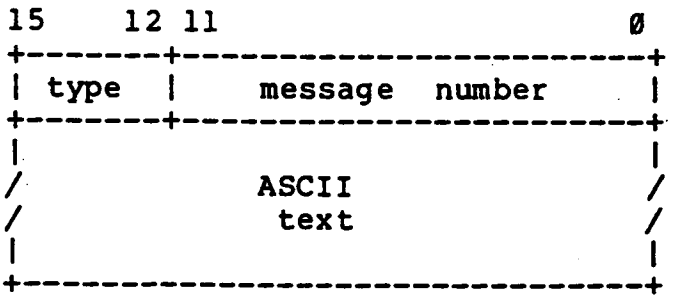
1. Remote programs written to interface to a terminal can be run transparently over a DUP connection.

2. The existence of a single Dialog Driver on the host side gives terminal users on that host access to all remote utilities and diagnostics utilizing the Standard DUP Dialog.

3. Host programs running without supervision by a user on a terminal need not analyze strings of ASCII text in order to hold up their end of the dialog if they are familiar with the remote program's input requirements.

The DUP Dialog is record-oriented and driven by the remote program. In order to embed this in the DUP protocol, the sequence of commands which the host program may issue must be restricted. The host program must issue a RECEIVE DATA with a byte count >= 80 upon receipt of the response to the command which initiated execution of the remote program. When the response to that RECEIVE DATA is received, the host program examines a type field in the received data and, depending on the value of the field, issues either another RECEIVE DATA or a SEND DATA followed by a RECEIVE DATA.

The format of messages from the remote program to the host program, which appear in the buffer specified in RECEIVE DATA commands, is as follows:



The ASCII text from the remote program to the host may include printable characters, including space, and the carriage return-line feed combination ONLY. No non-printable characters, escape sequences, or control characters are allowed.

Where Type is one of the following codes:

TYPE

1 QUESTION - The ASCII text is a prompt for information. The host program must issue a SEND DATA with the answer to the question in a form that the remote program understands. The message number uniquely identifies the question and the content and format of the answer to host programs which know the characteristics of the remote program. The host response must be in ASCII.

2 DEFAULT QUESTION - The DEFAULT QUESTION message is identical to the QUESTION message except that a null (zero-length) SEND DATA is taken to be a default answer to the question. If the host program is answering questions via message number and does not recognize the message number of a DEFAULT QUESTION message, it should respond with a null SEND DATA as its answer. This message type allows remote programs to add new capabilities requiring new user input parameters and still communicate with host programs which know only about an old version. The host response must be in ASCII.

3 INFORMATION - The ASCII text is an informative message. The message number uniquely identifies the type of information being transmitted. The INFORMATION message has its own message number space. The host program should issue another RECEIVE DATA command.

4 TERMINATION - The ASCII text is a normal termination message. The message number uniquely identifies the type of information being transmitted. The TERMINATION message has its own message number space. No further SEND DATA or RECEIVE DATA commands should be issued. Message number 1 is reserved to mean "simple termination" and does not have any ASCII text. Minimal-memory remote programs may omit the ASCII text on all TERMINATION messages.

5 FATAL ERROR - The ASCII text is a fatal error message. The FATAL ERROR message has its own message number space. No further SEND DATA or RECEIVE DATA commands should be issued. Minimal-memory remote programs may omit the ASCII text on all ERROR messages.

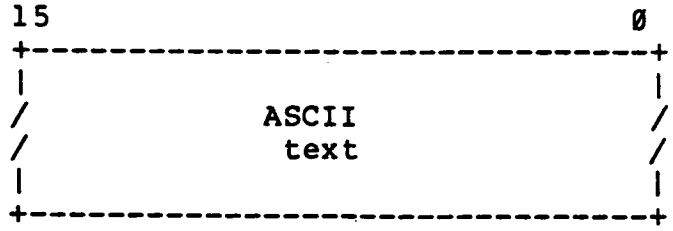
6 SPECIAL - This type is used when only a host program could respond. The message number indicates the type of special message. The data field is type dependent and not necessarily ASCII text.

Note that use of this message type is program dependent. Reference the remote program's functional specification. This message type should ONLY be used in cases where ONLY a program could respond.

Types Ø and 7-15 are reserved.

Message numbers are unique within type for a given program. Other programs may have the same message numbers.

The format of messages from the host program to the remote program, which are placed in the buffer specified in SEND DATA commands, is as follows:



APPENDIX A

MODIFIER CODES/RESPONSE STATUS CODES/OPCODES

The following are the opcodes for DUP commands:

GET DUST STATUS	-	1
EXECUTE SUPPLIED PROGRAM	-	2
EXECUTE LOCAL PROGRAM	-	3
SEND DATA	-	4
RECEIVE DATA	-	5
ABORT PROGRAM	-	6

The following are the modifier codes for DUP commands:

ALLOW STANDALONE	-	1 (bit Ø)

The following are the status codes for DUP responses:

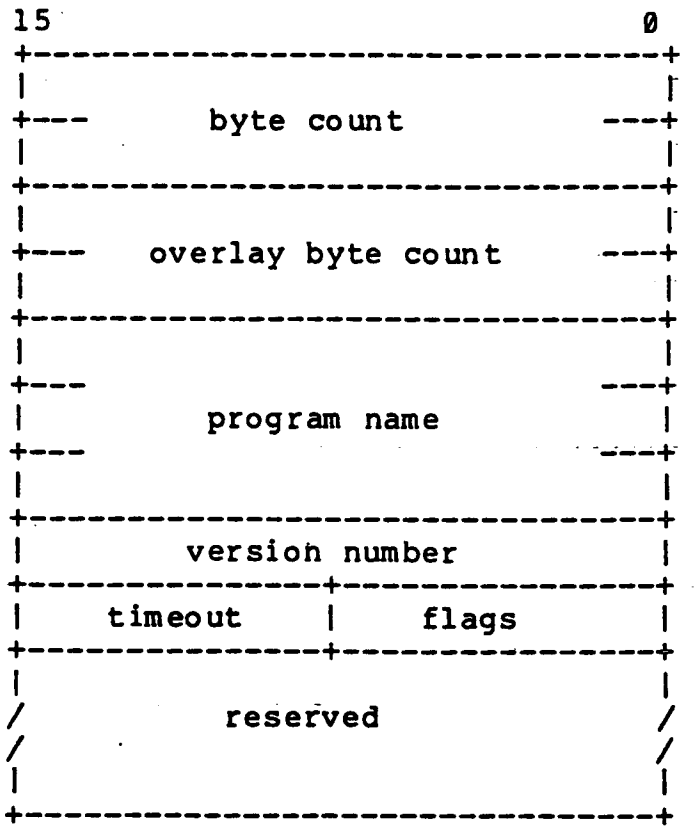
SUCCESSFUL	-	Ø
INVALID COMMAND	-	1
NO REGION AVAILABLE	-	2
NO REGION SUITABLE	-	3
PROGRAM NOT KNOWN	-	4
LOAD FAILURE	-	5
STANDALONE	-	6
HOST BUFFER ACCESS ERROR	-	9

The response codes returned by the controller have a value of 2ØØ octal plus the value of the opcode in the originating command. The only exception is that the response to an invalid command may have a value of 2ØØ octal instead of 2ØØ plus the originating command opcode. This is controller dependent.

APPENDIX B

REMOTE PROGRAM HEADER

Remote program images may reside either in the host or on media local to the controller. The host program may obtain certain of the characteristics of these programs in order to aid it in starting their execution and communicating with them when they are running. In the case of controller-local programs some of these characteristics are passed to the host in the response to the EXECUTE LOCAL PROGRAM command. In the case of host-resident program images, the program image starts with a fixed set of bytes, the program header, which gives the program's characteristics as follows. This header is added to the program via a host-dependent mechanism. The information in this header is used by the host to issue the EXECUTE SUPPLIED PROGRAM command.



byte count

Number of bytes in initial load plus the program header. The initial load image immediately follows the program header in the program image. The initial load image may contain an extended header with DUP-specific information.

overlay byte count

Number of bytes in overlay area. The overlay area immediately follows the initial load image in the program image.

program name

Program name (ASCII), space-filled on right

version number

Program version number, binary

flags

Bit Ø Set if this program is standalone.

Bit 1 If this bit is set, the program needs overlays from host memory during execution. The host must pass a second Buffer Descriptor describing the overlay buffer as part of the EXECUTE SUPPLIED PROGRAM packet.

Bit 2 If this bit is set, the overlay buffer should be writeable as well as readable from the controller.

Bit 3 Set if this program uses the standard DUP dialogue embedded within the SEND/RECEIVE DATA messages to communicate with the host. (see Section 5.Ø)

Bits 4-7 Reserved

timeout

Timeout value for SEND DATA and RECEIVE DATA commands in seconds. If Ø, these commands are not timed out.

reserved

Reserved. These bytes are reserved for future use.

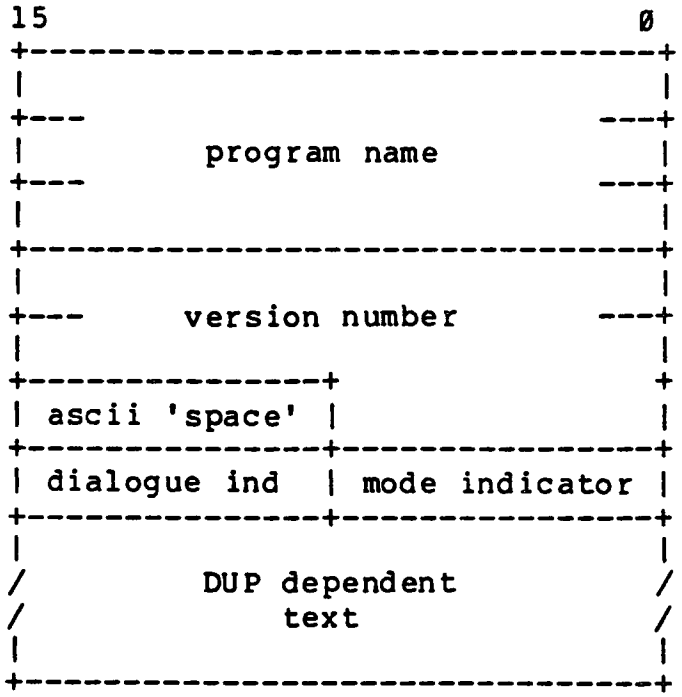
The EXECUTE SUPPLIED PROGRAM feature of DUP is provided to load utilities and diagnostics from the host into controllers which do not have enough local media to store them. Remote programs should be kept in a fixed directory within the host system. The extension of those down-line load files which will run on a given controller is specified

in the GET DUST STATUS Response.

APPENDIX C

THE DIRECT PROGRAM

DUP servers which desire to provide host programs with a directory of local programs should do so using a local program named DIRECT which uses the standard DUP Dialogue to report the directory. The directory is reported as a series of INFORMATION messages, all with a message number of 1 of the following form:



program name

program name (in ascii), right-filled with spaces

version number

Program version number (in ascii), left-filled with spaces

Space

ASCII blank

mode indicator

ASCII 'S' if program is STANDALONE, otherwise and ASCII space

dialogue indicator

ASCII 'D' if program uses standard DUP Dialogue, else ' '

DUP dependent text

Extra DUP-dependent informative text.

These INFORMATION messages may have other INFORMATION messages interspersed with them as long as those other messages have a message number other than 1. The stream of messages should end with a TERMINATION message with a message number of 1.