PROPOSED SMPTE STANDARD

SMPTE 273M

for Television —

Status Monitoring and Diagnostics Protocol

1 General

This standard defines the status monitoring and diagnostics protocol (SMDP) used with a general-purpose communication link that connects to equipment used in the production, post-production, and/or transmission of visual and audio information. The communication link is separate and distinct from the supervisory interface for digital control (ANSI/SMPTE 207M). SMDP may be used when querying equipment for status and diagnostics information.

The primary intent of this standard is to establish the protocol between a supervisory controller and associated video/audio equipment. In addition, it seeks to establish a common set of commands that should be used in SMDP-based systems. A provision allows for manufacturer-specific commands in addition to the common set.

2 Normative reference

The following standard contains provisions which, through reference in this text, constitute provisions of this standard. At the time of publication, the edition indicated was valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent edition of the standard indicated below.

SMPTE RP 165-1993, Error Detection Checkwords and Status Flags for Use in Bit-Serial Digital Interfaces for Television

3 Definitions

3.1 *encapsulated SMDP:* The lowest layer of the SMDP may be encapsulated as data into other protocols. Using this method, SMDP packets may be transported in open, standard protocols such as TCP/IP.

3.2 message set: From the standpoint of a sender, an issued command/data message sent to a destination and the associated response/data message from the destination. All message transactions between a supervisor and a virtual machine require a message set.

3.3 supervisor: Each system needs at least one supervisory controller. This controller connects to any and all video/audio equipment that is to be monitored.

3.4 *virtual circuit:* A transparent, unidirectional, logical communications connection between a supervisor and a virtual machine. The communication path may pass through other levels of protocol before the circuit is established.

3.5 virtual machine (VM): A logical device consisting of a single device or combination of devices that respond in essence as a generic type of equipment; e.g., VTR, video switcher, router, telecine, etc. The VM is the device under scrutiny when connected to a link that uses the SMDP.

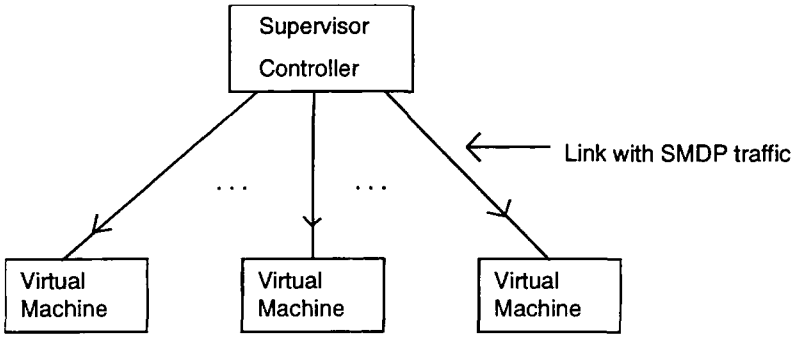


Figure 1 – Typical system

4 System elements

Figure 1 shows a typical system with links between a supervisory controller and several VM devices. Each device is connected to the supervisor using dedicated communication links (ANSI/EAI/TIA 232-E). Small systems may use directly connected communication links as shown in figure 1. Medium and larger systems may use encapsulated SMDP to provide addressibility, guaranteed delivery, and access to public networks. Encapsulated SMDP leverages the advantage of open, public network standards. Although SMDP may be used as shown in figure 1, its main application is as described in annex A. Annex A of this standard gives an example of encapsulated SMDP and how it may be used in a networked environment. As will be shown, SMDP traffic uses only a few ASCII control codes and, as such, may be carried by MODEMS without problems. The SMDP supports multidrop addressing. A discussion of multidrop modes is presented in annex B.

In general, the supervisor queries the VM using a command from the standard set or one provided by the manufacturer. The VM executes the command and always returns a response when finished. The VM may initiate a call to the supervisor by sending a special command for this purpose. So, the VM may be polled by the supervisor for information or it may send an interrupt to the supervisor indicating that it has information of interest. Most VMs also have a separate digital control interface input to remotely control the VM. It is the responsibility of the manufacturer to assign priorities to the two interfaces so that both are served but in a time and manner that is best suited for any particular VM.

SMDP traffic is processed by software routines (the SMDP processor) in each VM and the supervisor. The SMDP processor may be run as a task under the auspices of the VM's or supervisor's operating system. The processor is not multitasking and as such can only process one command at a time. The only exception is the *RST (reset) command which is executed upon arrival by the SMDP processor (see 6.1 for more information on *RST).

5 Common command and response structure

The SMDP defines 11 common commands and response messages that all protocol compliant virtual machines should implement. In addition, there are five that are optional. These are general in nature and not device-specific in any way. In addition, manufacturers may add private commands as long as they follow the byte and packet level protocols defined in this standard.

Next, the common set is defined and explained. All these message names include an * as part of the name to identify it as a common command. Manufacturer supplied commands should omit the *. Also, any command that is a query for information must include a ? to identify the query operation. Every command that is issued to a VM causes it to send one of three possible responses back to the supervisor. The possible responses are (1) *ATN:OPC, (2) ATN:CMDERR, or (3) *ATN:QRESP (plus response data structure). The ATN prefix signifies that the VM desires the attention of the supervisor. The OPC response indicates that the OPeration is Complete (no errors either) and the VM may receive another command. The CMDERR response indicates that the last command issued to the VM was errored in some way. The QRESP response indicates that data corresponding to a Query RESPonse is being sent. The command and response names are case insensitive.

Whenever ASCII string data are returned to the supervisor, any given line should not be longer than 62 characters including the carriage return/line feed combination, CRLF. The CRLF set allows for the convenient printing of returned response data. All string lines, regardless of length, should end with an ASCII CRLF combination (HEX 0D 0A). As many lines as needed may be returned for a query response.

Commands, responses, and data are sent between devices using packets. In general, packets to and from a VM are composed as follows. The braces below are used to delimit optional parameters and are not part of the actual command format.

Simplified packet for an issued command with data:

command_keyword{:command_keyword...} {?}
{ parameter,parameter...} ; data field

Simplified packet for an issued response with data:

response_keyword{:response_keyword...}
{ parameter,parameter...} ; data field

Commands, responses, and optional parameters go into the command/response field. A command is one or more colon-separated command_keywords. Likewise for a response. Associated data structures, if any, go into the data field. Values within braces are optional. Generally, commands go from the supervisor to the VM and responses go from the VM to the supervisor, but there are exceptions. The keywords and parameters must be printable strings. The following should be observed:

1) Keywords may be concatenated if desired. This is a hierarchical method of command/response naming. Keywords must be separated by colons (:). No colon is needed after the last keyword.

2) Any optional parameters must be comma (,) separated. There is no trailing comma after the last parameter. There must be a space between the last keyword and any parameters to follow.

3) The command/response field is always terminated with a semicolon (;). No other embedded semicolons are allowed in the command/response field. A ? is appended to the compound command name if the command is a query.

The command/data and response/data form a message set. For every command/data message sent to a receiver, a response/data message is returned. The response is, in effect, a termination for the transaction.

Table 1 summarizes the message names used in SMDP. The classification is as follows: C indicates that the name is a command message to the VM. R indicates that the name is a response message from the VM, CSUP indicates the name is a command message to the supervisor from the VM. If a $ appears next to the indicator, then command implementation is optional.

6 Command messages to the VM

6.1 Command name: *RST (reset)

Description: Reset the SMDP processor in the VM upon receipt of this command. This causes a termination of any pending or existing VM SMDP operations and it returns to the idle state. All error and message queues are flushed. Any current packet transmissions to the supervisor are cancelled. However, the *ATN:OPC is sent in response. When the VM powers on, the SMDP processor should be reset to the idle state. If the VM is attached to a multidrop link, or equivalent, this command causes the VM to unaddress itself (see annex B for more information.)

Table 1 – Command summary

Command or response message name		Full name	Classification
1	*RST	Reset	C
2	*IDN?	Identify query	C
3	*TST	Test	C
4	*TST?	Test query	C
5	*FLAGS?	Flag query	C
6	*STATUS?	Status query	C
7	*CMDERR?	Command error query	C
8	*MSG?	Message register query	C
9	*PIPE	Pipe	$C
10	*UPLOAD?	Upload query	$C
11	*ADDSEL	Select subaddress for VM	$C
12	*ATN:OPC	Attention:Operation complete	R
13	*ATN:CMDERR	Attention:Command error	R
14	*ATN:QRESP	Attention:Query response	R
15	*ATN:PIPE	Attention:Pipe	$CSUP
16	*ATN	Attention:Supervisor	$CSUP

Response of the VM: Return the *ATN:OPC response when the reset is complete.

6.2 Command name: *IDN? (identify query)

Description: Query the VM for device-specific information. The format of the returned information is a list of parameter identifier names, a colon, and the actual string parameter followed by the carriage return and line feed combination (crlf). The parameters with capital letters are user-defined values. There are seven predefined return types as shown below. These must be returned with values for all seven parameters. Use a default string value of NONE when no parameter is defined. Also, users may add custom parameters as needed. Insert these starting after the last row. Some VMs may have several associated serial numbers (SERIALNUM) or software versions (VERSION). It is the responsibility of the manufacturer to clarify these possible ambiguities in the VM's user manuals. The return data structure order and form is as follows:

Manufacturer:NAME[crlf]
Model:MODEL[crlf]
Device ID:DEVICEID[crlf]
Serial number:SERIALNUM[crlf]
Software version:VERSION[crlf]
Virtual machine type:TYPE[crlf]
VM subaddress:SUBADD[crlf]
(User defined parameters start here.)

A few of the parameters require some added explanation. The DEVICEID parameter is provided by the VM as a unique number or name identifier for that VM. The DEVICEID is useful for locating devices by name, but should not be considered an address for the VM. The manufacturer should define how to set the DEVICEID value for the VM. The virtual machine type parameter, TYPE, identifies what virtual machine control dialect is used by the VM. For example, SMPTE RP 170 gives the TYPE for video tape recorders as 02. TYPE values have been defined for several different classes of equipment by SMPTE. TYPE is a string value of a HEX number. The subaddress (SUBADD) for the VM defines which VM is selected when more than one device is attached to a single ANSI/EIA/TIA 232-E interface port. This situation may occur whenever a multidrop, or equivalent, configuration is implemented. If the VM is not configured in this fashion, use the string NONE as the address value (see 6.11 and annex B for more information.)

Response of the VM: Return the ASCII string *ATN:QRESP identifier immediately followed by the data structure, with parameters, shown above.

Example: A sample returned data structure from a VTR that was issued the *IDN? command.

Manufacturer:Video Things Inc [crlf]
Model:VTI-100 [crlf]
Device ID:VTR37 [crlf]
Serial number:1000512A [crlf]
Software version:2.1 [crlf]
Virtual machine type:02[crlf]
VM subaddress:NONE [crlf]

6.3 Command name: *TST TESTNUM (test)

Description: This command instructs the VM to execute numeric string TESTNUM. The test is run according to the manufacturer's specification for that test. Test number 0 is reserved. Tests range from 1 to FFFFFF. Test number 1 instructs the VM to perform a power-on test if one exists. All other tests are user defined. The results of each test are stored in a buffer that may be read by the *TST? command. The buffer's size is not specified by this standard.

Response of the VM: The VM returns a *ATN:OPC when the requested test has been executed.

Example: *TST 23 (Run test number 23 and put the results in the VM's test results buffer)

6.4 Command name: *TST? (test query)

Description: A query command to read the test results buffer (see *TST command). This buffer contains printable ASCII characters that describe the results from previously executed tests. The test results are retrieved in a first executed, first returned basis. If the result buffer overflows as a consequence of executing too many tests, the lest test's results are lost. The only way to empty the results buffer is by reading it using *TST? or issue a *RST. The actual returned data is a string (RESULT) containing the results. The string is preceded by the test number. If more than one line is needed, separate lines with a crlf combination.

Test:TESTNUM:RESULT [crlf]

Response of the VM: The response is in the form *ATN:QRESP + data structure. If there are no test results in the buffer, the returned data structure is: Test:0[crlf]. If several different test results are in the buffer, each time *TST? is executed, the results buffer will empty another response set until finally the Test:0 response is returned.

Example: Assume that test numbers 4 and 7 (used here by example only) have been successfully executed using *TST. Then issuing *TST? will return the data structure (RESULT).

Test:4:Power supply voltages are within range. VCC = 4.95, V12 = 11.8 [crlf]

By issuing *TST? a second time, the returned data structure could be

Test:7:Dynamic memory test passed. 512 kbytes tested. [crlf]

A third (or more) issuing of *TST? will return

Test:0 [crlf] (No buffered test results to return)

6.5 Command name: *FLAGS? (flag query)

Description: This query returns five flags from the VM. Each flag is a string yes or no with an associated flag description. The returned format is

Definition:{ yes no}

Some flags are "sticky," that is, once the flag is read it resets to "no," the default state. All five flags are returned per query. A crlf combination is used to separate lines. Additional user-defined flags may be added after the last line. The flags are a quick way to get a simplified picture of certain VM internal states. For more information, the message register (see *MSG?) could be used.

Definition of flags

The returned response from a VM should take the following format. The braces indicate a choice of one selection of the string values "yes" or "no."

Power cycled on: {yes,no} [crlf]
EDH now: {yes,no} [crlf]
EDH in past: {yes,no} [crlf]
EDA now: {yes,no} [crlf]
EDA in past: {yes,no}[crlf]

Power cycled on flag: Whenever the device power is cycled, this flag is set to "yes." This is a sticky flag. So, reading it will reset it to "no."

The following four definitions require SMPTE RP 165 to be observed. If not, set the flag to "no."

EDH now flag: This flag corresponds to the error detected here flag for the full-field checkword according to SMPTE RP 165. If a full-field checkword mismatch is currently occurring (during the field when the VM is queried), then this bit is set to "yes," else it is "no." If there is more than one digital video input that obeys SMPTE RP 165, the EDH now flag is set to "yes" if *any* of the inputs are generating an EDH fault.

EDH in past flag: If the "EDH now" flag was set at some time in the past (since power on of VM), this flag indicates so by being set to "yes." This flag is sticky.

EDA now flag: This flag corresponds to the error detected already flag in SMPTE RP 165. This flag is a "yes" if the received full-field EDA bit is a logic 1 at the time of the query of the VM. If there is more than one digital video input that obeys SMPTE RP 165, the EDA now flag is set to "yes" if *any* of the inputs have an EDA bit set.

EDA in past flag: If the "EDA now" flag was set at some time in the past (since power on of VM), this flag indicates so by being set to "yes." This flag is sticky.

User-defined flags: The manufacturer may define additional flags as needed to be appended after the fifth flag.

A received *RST command causes all flags to be set to "no."

Response of the VM: The VM returns the *ATN:QRESP identifier and the data structure below.

Example: The returned data structure could look as follows:

Power cycled on: yes [crlf]
EDH now: no [crlf]
EDH in past: yes [crlf]
EDA now: no [crlf]
EDA in past: no [crlf]

A second *FLAGS? query would return the following strings, assuming no changes in EDA or EDH activity between queries. Note the changed state of the sticky flags.

Power cycled on: no [crlf]
EDH now: no [crlf]
EDH in past: no [crlf]
EDA now: no [crlf]
EDA in past: no [crlf]

6.6 Command name: *STATUS? (status query)

Description: This is a query for the current operational status of the VM. The result is a printable ASCII string indicating the status. This command should not be confused with the *MSG? command described below. The status information returned by *STATUS? is defined by the manufacturer and should be the predominate operational condition of the VM.

Response of the VM: The VM returns the *ATN:QRESP indicator plus the data structure. The data structure is a string.

Example: A possible returned string could be: VTR in play [crlf]

6.7 Command name: *CMDERR? (command error query)

Description: Every command sent to the VM should be checked for syntax, and parameter errors. If such errors occur, the VM loads the command error buffer with an error string indication. The VM command error buffer holds error strings in a queue and empties successive strings in response to each *CMDERR? received. The buffer is emptied in a first set, first read order. Once the last non-zero error code has been read, error code zero is returned for any further responses. The return codes are defined as given in table 2.

The format of the returned data structure is

Command in error -> error code:error description [crlf]

or by exception

*CMDERR -> No errors in queue [crlf] (No errors to return)

Table 2 – Error code assignment

Error code	Error description
0	No errors in queue
1	Syntax error. Command not recognized
2	Syntax error. Parameter out of limits or unexpected type
3	Syntax error. Too few or too many parameters
4–10	Reserved
11	Flow control value out of range
12	Data type value out of range
13–20	Reserved
21	Packet error: Received packet had too many bytes
22	Command not implemented yet
23	Command not executed. Insufficient memory
24–100	Reserved
101–999999	User-defined error codes

Response of the VM: The VM returns the identifier *ATN:QRESP plus the next error in the queue in the format just described.

Example: Say the misspelled command string *TSST 4 was sent to a VM. The VM would respond with a *ATN-CMDERR indicating that an error was detected. The supervisor could now respond with a *CMDERR? to retrieve the actual error message. The VM response data would be

*TSST 4 -> 1:Syntax error. Command not recognized [crlf]

So, it is obvious TST was misspelled. Now, if the VM's error queue were empty, other queries of it would return

*CMDERR -> No errors in queue [crlf] (This is a special case. No command as a prefix.)

6.8 Command name: *MSG? REGNUM (message register query)

Description: This command queries the message register set for data strings. The register set is a group of registers that contain information relating to the health and operation of the VM. Each row of the compound register array contains a printable string. The message registers may be thought of as a proxy for the front panel indicators of the VM. Most information on a traditional front panel may be included in the data structures of the message registers if desired. The registers are defined as shown in table 3. All messages are user defined. Three types of information are available, namely warnings, errors, and status. REGNUM is a string number from HEX 0 to FFFFFF.

Table 3 – Message registers

Register number	Description
0	Pointer, pointer, ..., pointer to active warnings/errors
000001-00FFFF	User error responses
010000-01FFFF	User warning responses
020000-02FFFF	User status conditions
030000	Event trigger condition for *ATN
030001-0FFFFF	Reserved
100000-10FFFF	Reserved error responses
110000-11FFFF	Reserved warning responses
120000-12FFFF	Reserved status conditions
130000-FFFFFF	Reserved

Register 0 is reserved for pointers to active error and warning registers. The status registers are not pointed to. The pointers are string register numbers separated by commas. The returned string (from any register) may be of any length with crlf inserted as needed so returned lines are < = 60 characters. It is the responsibility of the VM to set and maintain the accuracy of

register 0 and those registers to which register 0 points. It is a good practice for the supervisor to read register 0 first to identify what registers have warnings or errors in them.

Register zero never points to any status condition registers. It is the responsibility of the manufacturer to document the registers that are used for all three types of data. Warnings and errors are distinguished by the nature of the message. It is left to the manufacturer to decide what messages are labeled warnings and errors. Also, when a row is requested that is undefined, inactive, or reserved, the response data should be the unquoted string "not active [crlf]" unless defined differently by the manufacturer. If no errors or warnings exist, a query of register zero should return: 0 [crlf]. Register 30000 is set aside for recording trigger conditions related to the *ATN command (see 8.1).

Response of the VM: The VM returns the *ATN:QRESP response identifier and a string data structure defined by the manufacturer.

Example: Assume the VM had a message register set as shown in table 4.

If the supervisor issued a *MSG? 0, the returned response data would be: 3,10001 [crlf]. This response indicates one error and one warning exist. Next, issuing *MSG? 10001 would return the string response data: 1000 head hours reached [crlf]. Register 0 indicates that registers 3 and 10001 contain messages. Notice that register 0 does not point to any status information. Status registers should be defined by user documentation.

Table 4 – Example of message register contents

Register number	Description
0	3,10001 [crlf] (pointers)
....	....
3	Tape tension error [crlf] (error)
10001	1000 head hours reached [crlf] (warning)
20001	Power supply OK [crlf] (status)
20002	VTR in play mode [crlf] (status)

6.9 Command name: *PIPE TYPE (pipe)

Command format: This command must use a type 1 packet with flow = 0 (see clause 9 for more information on the packet structure of SMDP messages). Any data that is passed between peer entities may be in string or binary format and loaded into the data field of the packet.

Description: This message identifies a unidirectional virtual connection between the supervisor and the VM. The pipe path allows for a supervisor process to send string/binary data to a peer process at the VM side of the connection. User defined string TYPE identifies both the VM process for which to pass the data structure and the protocol of the passed data. The purpose of the PIPE command is to allow for other non-SMDP message protocols to be passed between the processes at the two ends of the pipe (see the *ATN:PIPE command for a discussion of the virtual connection from the VM to the supervisor). This command is optionally implemented. No *ATN:OPC message is sent in response to the *PIPE command. This is an exception to the general rule. The data structure associated with this command is completely user- defined. The usage of the *ATN:PIPE and *PIPE commands are described in annex C.

6.10 Command name: *UPLOAD? TYPE{,size} (upload)

Description: This command is used to upload a binary block of data from the VM. The data that is uploaded into the supervisor may take many forms. For one, it may be graphical user interface (GUI) application binary object code for use by the supervisor as a user interface to the VM. Or, it may be a help file. Or, it may be a list of nonstandard commands that the VM supports. This standard does not define the usage or size restrictions for the uploaded data. The user manual for the VM should indicate exactly what data blocks are available, their size, and how to use them. TYPE is a string variable that identifies the data block to be uploaded. TYPE = BINARY is the only named block defined and is done so strictly for convenience. If other binary blocks exist, name them with TYPE names of the manufacturer's liking (see table 5).

Table 5 – TYPE definition

TYPE values	Description
Binary	Returns binary object code for use by the supervisor
Other TYPE names	User defined

SMPTE 273M

There is an optional parameter within the braces of the command. This is the unquoted string "size." When "size" is included in the command, the return is not a data block, but rather the size of the data block referenced by TYPE. So, if *UPLOAD? binary,size were issued then, the response data structure would be a string numeric indicating the size in bytes of the block of binary object code referenced by "binary."

Response of the VM: The VM returns the *ATN:QRESP identifier plus the block data structure.

For example, a supervisor issued command: *UPLOAD? binary,size could generate a string response of: 10004 [crlf]. This indicates that the block labeled "binary" has a size of 10,004 bytes. By issuing *UPLOAD? binary, the VM would return the *ATN:QRESP identifier followed with a binary block of 10,004 bytes.

6.11 Command name: *ADDSEL SUBADD, {on off} (VM subaddress select)

Description: This command is used to select (or unselect) one VM among a number of VMs that share a common SMDP interface port. Such a configuration is typical when devices share a link in a multidrop fashion (see annex B for more information). If VMs are not connected as described in the annex, then this command is not required.

The range value of SUBADD is numeric string 0 to FFFF hex (leading zeroes not required). SUBADD = 0 is reserved and should not be assigned to any VM. The assignment of subaddresses is manufacturer dependent and not defined by this standard. The SUBADD value is referred to as a subaddress since it is used to select among VMs attached to a single interface port. In addition to the subaddress parameter, there is a mode parameter. This parameter is selected from the choices within the braces. The possible modes are "on" or "off." Sending "on" to a VM indicates that the VM with SUBADD as a designated subaddress is to become active and will respond to any further commands from the supervisor. Likewise, sending the parameter "off" to the VM with subaddress SUBADD will cause it to become unselected. Selecting an already selected VM is allowable and should cause no side effects. Only one VM should be selected at any instant in time. Activating more than one VM simultaneously will yield unpredictable results.

Assuming a configuration similar to the figure in annex B, the *ADDSEL command may be used as shown in the following example of command sequences to a multidrop port:

*ADDSEL 02,on	(Select VM with SUBADD = 02)
*IDN?	(Ask for the *IDN response for example)
.....	(Other commands to 02)
*ADDSEL 02,off	(Unselect VM 02. This must be sent before any other VM is selected)
*ADDSEL 05,on	(Select VM with SUBADD = 05)
.....	(Other commands to 05)
*ADDSEL 05,off	(Unselect VM)

The example demonstrates several important points. First, commands are executed only by the selected VM. Also, only one VM should be active at any instant of time. Further, the selected VM should be unselected before another VM is activated.

Response of the VM: *The selected VM will return a* *ATN:OPC whenever it is selected or unselected by the *ADDSEL command.

7 Response messages from the VM

As mentioned, the VM must return one of three possible response messages for every received command. These three message types are outlined below. Messages are to be loaded in the command/response field of the sent packet. The ATN prefix indicates that the supervisor should give attention to this response.

7.1 Response message: *ATN:OPC (Attention: operation complete)

Description: The VM returns this message in the command/response field whenever the last nonquery command has been successfully executed. There are no associated data to be sent.

7.2 Response message: *ATN:QRESP + data structure (Attention: query response)

Description: *The VM returns this message in the* command/response field whenever the last issued command by the supervisor required a query response. The data structure immediately follows the *ATN:QRESP string and is in the data field of the same packet. Individual query commands define their respective returned data structures.

7.3 Response message: *ATN:CMDERR (Attention: command error)

Description: The VM returns this message in the command/response field whenever the last command had syntax or other problems as described in table 2 (see the *CMDERR? command). There is no associated data to be sent.

8 Command messages to the supervisor

There are two messages (commands) that the VM may initiate on its own. One is *ATN, the other is *ATN:PIPE. These are not VM response messages in the sense so far described in this standard.

8.1 Command name: *ATN (attention)

Description: The VM may send a wake-up message, *ATN (an interrupt in effect), to the supervisor under certain conditions. Typically, receipt of this command informs the supervisor that a new warning, error, or other condition has occurred. *ATN is not a required command and is optionally implemented. The enabling and disabling of trigger conditions for which *ATN is generated are not defined by this standard.

In addition to sending *ATN to the supervisor, the VM may load a textual description of the event trigger condition into the reserved message register number 30000 (see 6.8) whenever a new trigger event occurs. In practice, the supervisor may query register 30000 to view the trigger condition for which the last *ATN was sent to the supervisor. This event recording feature is optionally implemented. If register 30000 is not used or set, a query of it should return "not active (crlf)."

8.2 Command name: *ATN:PIPE TYPE + data structure (Attention:pipe)

Command format: This command must use a type 1 packet with flow = 0. Any data that are passed between peer entities may be in string or binary format and loaded into the data field of the packet. Packet type and flow are discussed later in this standard.

Description: This message identifies a unidirectional virtual connection between the VM and the supervisor. The pipe path allows for a VM process to send binary data to a process at the supervisor side of the connection. User defined string TYPE identifies both the supervisor process for which to pass the data structure and the protocol of the passed data. The purpose of the PIPE command is to allow for other non-SMDP messaging protocols to be passed between the processes at the two ends of the pipe (see the *PIPE command for a discussion of the virtual connection from the supervisor to the VM). No *ATN:OPC message is sent in response to the *PIPE command. This is an exception to the general rule. This command is optionally implemented. The data structure associated with this command is completely user defined. The usage of the *ATN:PIPE and *PIPE commands are described in annex C.

9 Data link layer protocol

The SMDP is packet based. Each packet is made up of bytes with the structure as shown below. The protocol is very compact with only four different packet types. These packet types are shown below. All message types are sent using packet types 1 to 4.

9.1 Byte definition

Start bit	B0 (LSB)	B1	B2	B3	B4	B5	B6 (MSB)	Parity (ODD)	Stop bit

Start bit = 1
Stop bit = 1
Data bits = 7
Parity = odd

9.2 Packet definition

Packet type 1 is used for commands, response messages, and data. Commands, responses, and most associated parameters go into the C/R (command/response) field. This field is always terminated with a semicolon (;). The C/R field length is a variable with the maximum length, including the ;, of 60 bytes. The terminating ; character should not be embedded as part of the command, response, or parameters. The packets all start and end with the ASCII SYN (sync) character. The C/R field, the two control bytes (flow, mode), and the data field are defined by the header pair of a SYN and the STX and the trailer pair of ETX and SYN. STX marks the start of text and ETX marks the end of text. The data field is of variable length up to a maximum of 512 bytes. The incoming packet holding buffer should be at least 580 bytes deep, the approximate maximum length of a type 1 packet.

Packet type 1 definition:

1 byte	1	max 60	1	1	max 512	1	1
SYN (16H)	STX (02H)	C/R;	Flow	Mode	Data	ETX (03H)	SYN (16H)

Packet type 2 is used whenever there are no data to be sent.

Packet type 2 definition:

1 byte	1	max 60	1	1
SYN (16H)	STX (02H)	C/R;	ETX (03H)	SYN (16H)

Of the 16 common command/response messages, all are sent using a type 2 packet except for *ATN:QRESP, *ATN:PIPE, and *PIPE which use a type 1 packet. The C/R field is limited, including embedded parameters, to ASCII characters in the range from 20H to 7FH. The data field is limited to the range 20H to 7FH, CR (0DH), and LF (0AH).

Packet types 3 and 4 (3 bytes each) are used for acknowledging receipt of good or bad packets. Also, the ACK packet is used to control the supervisor's or VM's SMDP buffer filling. In general, the packet receiver sends an ACK upon the complete receipt of a good (no byte level errors) packet and a NACK for the conditions in table 6. As soon as a received error is encountered and the NACK is returned, the sender must resend the last packet. Timeouts and limiting endless retransmissions are the responsibility of the supervisor.

Packet type 3
ACK packet

SYN (16H)	ACK (06H)	SYN (16H)

Packet type 4
NACK packet

SYN (16H)	NAK (15H)	SYN (16H)

Table 6 – ACK/NACK responses

Received packet condition	Acknowledgment packet from receiver
Byte parity error	NACK
Framing error (No stop bit recognized)	NACK
Overrun error (Input buffer overflow before VM could accept the next byte)	NACK
Received packet had no errors of the type above and receive buffer is ready for next packet	ACK

9.2.1 Flow control

The flow control byte, Flow, is used to regulate the flow of concatenated type 1 packets. Large data blocks must be split into smaller chunks so that the packet length does not overflow. For data lengths greater than 512 bytes, an additional packet(s) is required to send the data. Whenever a data block is fragmented into two or more packets (called a packet sequence), the flow byte is needed to regulate the packet sequence. Each packet in the sequence should bear the same C/R field message (see table 7).

Table 7 – Flow byte definition

Flow byte	Condition
ASCII 30H (numeric zero)	Last packet in sequence
ASCII 31H (numeric zero)	Next packet in sequence
Other	Reserved

9.2.2 Comment on packet and command control flow

Packets flowing between the supervisor and VM pair use the following packet control logic:

- Send a single packet to the destination
- Sender waits for ACK or NACK from destination
- If NACK, then resend packet
- If ACK and another packet is to be sent, then send next packet in packet sequence
- Continue until all packets are sent in sequence

The ACK response is used to regulate packet flow between the sender/destination pair. Without the regulation, the receive buffer could overflow.

On the other hand, the command flow is regulated by a termination response from the destination. As shown previously, every command sent has a corresponding response message. The next command should not be sent until the existing message set is complete. So, both packet flow and command flow have regulating mechanisms. The SMDP processor is not multitasking and, as such, commands are executed one at a time. The only exception is the *RST command. The SMDP processor should execute the command immediately on receipt.

9.2.3 Mode control

The mode byte identifies the type of data in the data field (see table 8).

Table 8 – Mode byte definition

Mode byte	Condition
ASCII 30H (numeric zero)	Data is string type. ASCII 20-7F Hex, OD and 0A (CR LF)
ASCII 31H (numeric zero)	Data is coded binary
Other	Reserved

9.2.4 Coded binary

The data field in a type 1 packet may carry coded binary if the mode byte indicates so. Since only 7-bit ASCII codes are allowed in the data field, 8-bit octets (range 0 to 255) may not directly be sent from source to destination. When the data to be sent are composed of 8-bit values, these target octets must be coded first into an intermediate ASCII code. It is this code that is placed into the packet data field.

The target octets are coded two bytes at a time. The coded binary result will be composed of three ASCII bytes. So, for every two octets of target data, three characters are placed into the data field of the packet. Each pair of targets is coded independent of any other target bytes.

Target byte 1	Target byte 2

These 2 target bytes are coded into the triplet of legal ASCII values:

x1	zz	x2

The values x1, zz, and x2 are coded ASCII. The coded values are loaded into the packet data field so that x1 is sent, then zz, then x2, and so on. Tables 9 and 10 are also needed. In these tables, Z1, Z2, X1, and X2 are decimal values. These values are the weights assigned to the coded ASCII values. Their corresponding coded values are indicated by a lower case parameter as shown in the tables.

Table 9 – ASCII code and corresponding coded weight for X1 and X2

x1 or x2 ASCII code		Decimal weight for X1 or X2
20 hex	(space)	0
21	(!)	1
22	(")	2
...		...
7E	(~)	94

See a complete ASCII table for the missing continuous values in table 9.

Table 10 – ASCII code and corresponding coded weight for Z1 and Z2

Decimal weight for Z1	Decimal weight for Z2	zz ASCII code	
0	0	31 hex	(1)
0	1	32	(2)
0	2	33	(3)
1	0	34	(4)
1	1	35	(5)
1	2	36	(6)
2	0	37	(7)
2	1	38	(8)
2	2	39	(9)

where zz = Function (Z1,Z2).

The following relations are used to code the target bytes to coded values:

Target byte 1 value = Z1*95 + X1 (1)

Target byte 2 value = Z2*95 + X2 (2)

where the target byte value is the decimal value of the 8-bit target byte from 0 to 255.

To code the two target bytes into three coded bytes, use the following rules:

- Find X1 and Z1 for target byte 1 using (1)
- Find X2 and Z2 for target byte 2 using (2)
- Find zz based on Z1 and Z2 using table 10
- Find x1 and x2 based on X1 and X2 using table 9

For the sake of an example, assume the target bytes, in decimal, are:

105	250

then, X1 = 10 and Z1 = 1 so that 95*1 + 10 = 105, and X2 = 60 and Z2 = 2 so that 95*2 + 60 = 250.

Using code tables 9 and 10, the three coded bytes are (with zz = numeric 6 since Z1 = 1 and Z2 = 2)

*	6	\

with * (hex 2A) as the code for X1, the 6 (hex 36) as the code for the combination of Z1 and Z2, and \ (hex 5C) for X2. So the actual three coded values sent in the outgoing packet are 2A 36 5C.

If there is an odd number of target bytes, treat the last byte alone, do not send the code byte for x2 and let Z2 = 0 by default. So, two code bytes are sent instead of three for this case. Since a type 1 packet is used when sending binary data, the ETX in the trailer may be sensed to indicate the case of an odd number of target bytes.

9.2.5 Examples of packet usage

The following examples demonstrate how packet types 1 and 2 may be used. Assume that the supervisor sends a *MSG? 3 command to the VM. The sent packet will look like

Packet type 2

SYN	STX	Command	ETX	SYN
16H	02H	*MSG? 3;	03H	16H

The VM responds with an ACK packet assuming no problems are detected. Since the command was a query, the VM will then send a packet type 1 with the data response. The *ATN:QRESP response message identifies the returned packet information as a query return.

Packet type 1

SYN	STX	Response	Flow	...
16H	02H	*ATN:QRESP;	30H	...

...	Mode	Data	ETX	SYN
...	30H	Tape tension error [crlf]	03H	16H

The supervisor responds by sending an ACK to the VM if the packet was received in good condition. This terminates the packet exchange between the two sides until another transaction occurs. If the returned data was too long for the 512-byte data field, then the VM should break up the return into two or more packets. The flow control byte is needed for packet sequences greater than one packet.

10 State diagram for SMDP processing

Figure 2, the state diagram for the SMDP processor, will now be described. Specifically, the state diagram for the VM will be given. It is intended to show how the VM processes input/output bytes on the SMDP interface. The supervisor has a similar state description and is not described in this standard. The SMDP processor is usually governed by a higher level VM process or the VM's operating system. As such, the SMDP processor may be preempted by a higher level software process at any time.

In order to understand the state diagram shown in figure 2, a few definitions will be given. All possible transitions between states are represented by arrows between the states. Each transition is qualified by an expression (exp) on individual arrows. Additionally, transmitted messages over the link are expressed by T(msg) while received messages from the link by R(msg). Also, whenever a data structure is too long to be sent in one packet, it must be split into two or more packets. This is called a packet sequence and is referenced in the state diagram. Associated with figure 2 are several notes:

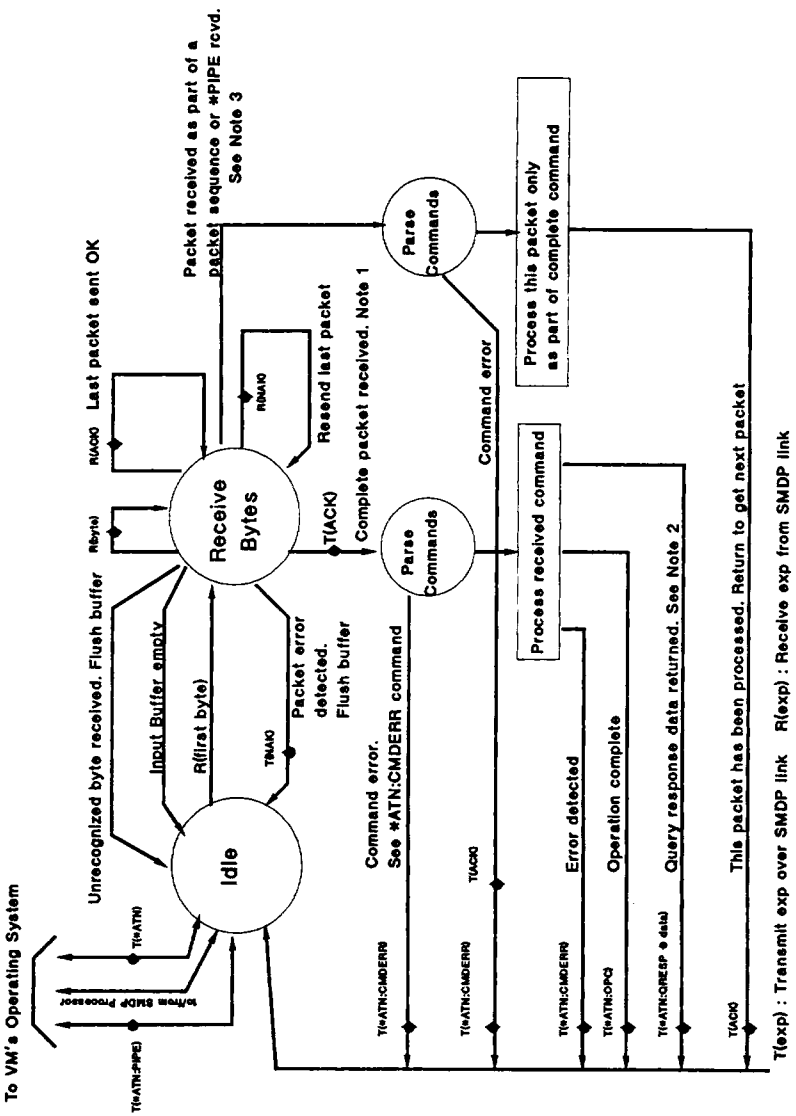


Figure 2 – SMDP processor state diagram of VM

NOTES

1 Command received in a single packet. No other packets to follow as part of this command. Or, the last packet is being received as part of a packet sequence.

2 For this query response, one or more packets may be returned depending upon the length of data sent. If a packet sequence is to be returned, the VM should follow ACK/NAK protocol for each packet. The ACK/NAK protocol for sending a packet sequence is not shown on this state diagram. However, the logic is similar to that for receiving a packet sequence which is shown on this diagram.

3 A packet was received that is part of a packet sequence, but not the last packet in that sequence. Process this packet and expect at least one additional packet in the sequence to follow. Or, the *PIPE message was received. This message is a special case in terms of processing since *PIPE is treated differently from other commands.

11 Physical port description

If a VM that supports SMDP uses a separate connector for transporting the protocol, then the link definition is as follows:

The VM always acts like data terminal equipment (DTE). ANSI/EIA/TIA 232-E as used by VMs supporting SMDP must function with at least a bit rate of 19.2 kbit/s with 7 bits, odd parity. Other speeds are permissible both higher or lower than 19.2 kbit/s as defined by the manufacturer. Two connector types are allowed with external cable lengths limited only by the ANSI/EIA/TIA 232-E specifications.

A 25-pin DB25 female is connected as shown in Table 11.

Table 11 – 25-pin DB25 female connections

Pin number	Signal name
1	FG frame ground
2	TXD transmit data (output)
3	RXD receive data (input)
4	RTS request to send (output)
5	CTS clear to send (input)
6	Unused
7	SG signal ground
8	Unused
20	Unused

A 9-pin female connector may also be used. The connections are as shown in table 12.

Table 12 – Nine-pin female connections

Pin number	Signal name
1	Unused
2	RXD receive data (input)
3	TXD transmit data (output)
4	Unused
5	SG signal ground
6	Unused
7	RTS request to send (output)
8	CTS clear to send (input)
9	Unused

11 Guidelines

The following are some general guidelines that should be observed:

1) The receiver must be able to accept the incoming data bytes at the fastest allowed link rate. If the VM is busy with other more pressing business, it still has the responsibility to buffer new bytes up to the buffer limit. If not, a buffer overrun error may be encountered and force the retransmission of the last packet.

2) Response timeout management is the responsibility of the supervisor. No timeouts are specified in this standard. Whenever the SMDP packets are encapsulated into a LAN or WAN network, timeout management should reflect this fact.

3) The SMDP uses only byte parity for the checking of bad packet data. When SMDP is encapsulated by higher level protocols, these protocols will provide for error checking.

4) Some tests and operations executed by the SMDP processor may interfere with the normal operation of the VM. If, for example, a video tape machine is recording and the SMDP processor attempts to execute a test that disables the record process, unexpected results will likely occur. It is the responsibility of the user performing diagnostics operations to avoid conflicts of this nature.

Annex A (informative)
Encapsulating SMDP packets in an open network protocol

This annex explains how the SMDP packets may be encapsulated into open network standards. For medium to large SMDP systems, encapsulation of packets offers great benefits. For one, the SMDP communication link may be LAN or WAN based and not confined to point-to-point wiring. Protocols that may be described by the open systems interconnect (OSI model) layers offer addressing and segmentation control. This unburdens the SMDP from the same tasks. Using open LANs allows the benefit of both SMDP and open systems to be achieved together. Hundreds, or even thousands, or widely separated VMs may be managed by a single supervisor with this concept. Multiple supervisors must negotiate between themselves for control.

Exactly what method is used to encapsulate SMDP is left to those who implement the network. Figure A.1 is an example of a method that is practical using encapsulation. This standard does not endorse one method over another.

This example uses TCP/IP (transmission control protocol/internet protocol) and a service called TELNET to carry SMDP packets across a network. This annex is not meant to be a tutorial on TCP/IP, TELNET, or LAN (see figure A.1 for a possible network layout).

The supervisor has access to all VMs on the network. The LAN may be any variety of Ethernet, FDDI, or other LAN that supports TCP/IP. The terminal server, a commercially available device, does the following functions:

- LAN interface
- ANSI/EIA/TIA 232-E output to VM
- TELNET service built within server
- Buffering, addressable output ports using TELNET
- One LAN port to many ANSI/EIA/TIA 232-E ports (1 to 100 is possible)

How do the SMDP packets get transported between the supervisor and the terminal server? The TCP packets on the LAN carry SMDP packets as a payload. With this arrangement, SMDP packets are sent across the LAN in another protocol's data field. It is very simple and allows for leveraging the strengths of well-established protocols. Figure A.2 shows how, in simplified form, SMDP may be encapsulated.

Notice how SMDP is just data in another protocol's data field. Using TELNET, a type of pipe between a receiver and a transmitter, SMDP is easily transported to and from the supervisor/terminal server pair. By issuing a TELNET call, the supervisor can send SMDP packets to the terminal server over the LAN. The VM does not need a TELNET service, but just returns its response on the SMDP transmit port. The terminal server is responsible for encapsulating these SMDP packets and forwarding them to the supervisor.

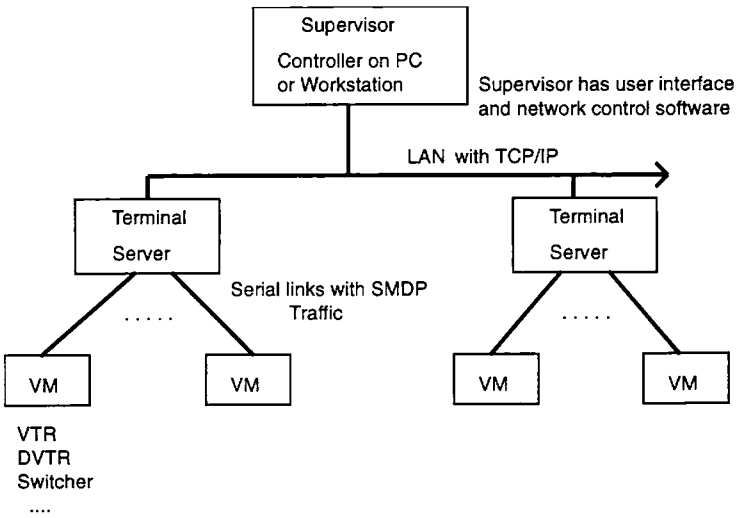
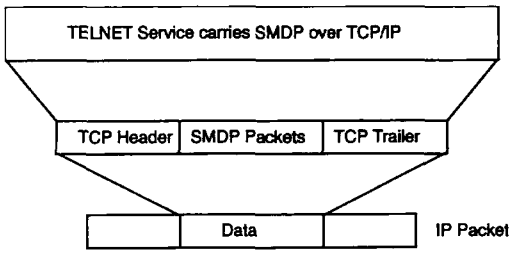


Figure A.1 – Possible network layout



Carrying SMDP using TELNET and TCP/IP

Figure A.2 – SMDP encapsulation

Annex B (informative)
Multidrop configurations

This annex describes how SMDP is used when more than one VM is connected to a multidrop system. Under non-multidrop operation, *the SMDP has no addressing inherent* in its protocol. When a multidrop configuration is required, then a form of addressing (called subaddressing) is needed. Figures B.1 and B.2 show possible multidrop systems.

In both cases, more than one VM is attached to a bus. In order for the supervisor to converse with, say, VM 03, the SMDP must have a way to address 03 (the subaddress) and enable its port on the tri-state bus. By using the *ADDSEL command with the "on" option, the supervisor can select *SUBADD = 03 and thus initiate a dialog with VM 03. All other* VMs on the bus should be disabled except for the selected one (see the *ADDSEL command in 6.11 for definitions and examples of usage). The supervisor may unselect 03 at any time using the "off" option in the *ADDSEL command. Any commands sent to unselected VMs will not be responded to. By proper use of the *ADDSEL command, any single VM may be enabled for dialog with the supervisor.

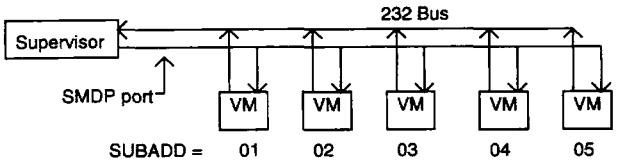


Figure B.1 – ANSI/EIA/TIA 232-E bus with tri-state drivers

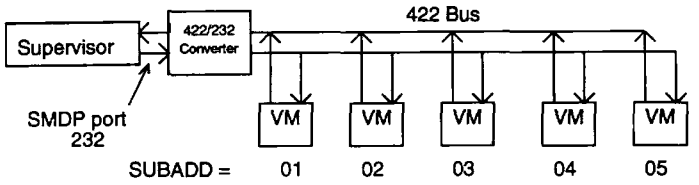


Figure B.2 – EIA-422-A bus with tri-state drivers and converter interface

Annex C (informative)
Using the PIPE commands

The PIPE commands (*PIPE TYPE and *ATN:PIPE TYPE) are useful when another messaging protocol needs to be transported between the supervisor and VM. The intent of PIPE is to allow for uninterpreted (by SMDP) binary data to be linked between two processes. Both the supervisor and the VM should contain a process (referenced by TYPE) at both ends of the pipe. Figure C.1 shows the flow of data.

When the SMDP receiver recognizes the PIPE command, it passes the data in the packet(s) to the process specified by TYPE. For a bidirectional link between the two peer processes, the supervisor uses *PIPE to send data and the VM uses *ATN:PIPE to send data. By using PIPE, devices may receive and send various messaging protocols, like those referenced by ESbus or ESlan, for example. It is the responsibility of the VM to guarantee that the piped protocol is handled correctly in every respect. There is no limit to the amount of data that is transferred across a pipe other than those imposed by the VM itself. The transfer rates of data must be within those allowed by SMDP, as well. Data sent with the pipe commands are undefined by this standard.

By way of example, say the supervisor sends an ESbus message, REWIND, to a VTR. If the receiving process at the VM is identified by TYPE = ESBUS and is capable of executing ESbus commands, then the following packet could be sent using *PIPE:

1 byte	1	Command field	Flow	
SYN (16H)	STX (02H)	*PIPE ESBUS;	0 (30H)	...

	Mode	Data field	1	1
...	0 (30H)	REWIND	ETX (03H)	SYN (16H)

Likewise, the VTR could respond with:

1 byte	1	Response field	Flow	
SYN (16H)	STX (02H)	*ATN:PIPE ESBUS;	0 (30H)	...

	Mode	Data field	1	1
...	0 (30H)	REWINDING	ETX (03H)	SYN (16H)

Other configurations besides the two shown above are possible. For example, it is possible to imagine embedded VMs within another VM. For this case, any embedded VM may be enabled using the *ADDSEL command assuming the manufacturer provides for it. It is left to the manufacturer of the VM to define how subaddresses are set and maintained.

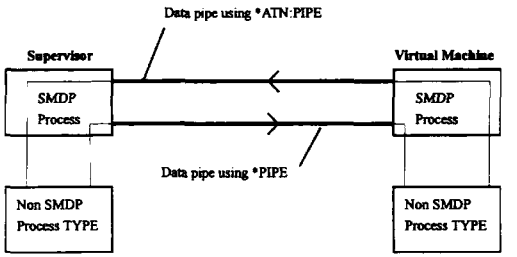


Figure C.1 – Data flow

Annex D (informative)
Bibliography

ANSI/EIA/TIA 232-E-1991, Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange

ANSI/SMPTE 207M-1992. Television — Digital Control Interface — Electrical and Mechanical Characteristics

EIA-422-A-1978, Electrical Characteristics of Balanced Voltage Digital Interface Circuits

SMPTE RP 113-1992, Supervisory Protocol for Digital Control Interface

SMPTE RP 138-1992, Control Message Architecture

SMPTE RP 139-1992, Tributary Interconnection

SMPTE RP 170-1993, Video Tape Recorder Type-Specific Messages for Digital Control Interface

SMPTE EG 29-1993, Remote Control of Television Equipment