REFERENCE MATERIALS

The *Reference Materials RM1–4 and RM5–16* provide reference information for the user of the PS 390 system. Summaries of the ASCII commands, intrinsic functions, initial function instances and GSRs are contained in the first part of the volume. Included in the second part of the volume are sections covering interactive devices, interfaces and options, host input data flow, system function network diagrams, diagnostic utilities, system errors and host communications. The final section contains an index to the complete *PS 390 Document Set.*

RM5 Host Communications

This section includes descriptions of the RS-232 specifications and pin connector definitions, PS 390 transmission protocol, port values and defaults, and the PS 390 system data reception functions.

RM6 Interfaces / Options

This section contains information about the PS 390/host interfaces.

RM7 Host Input Data Flow

This section covers information on the host input data flow, including routing functions and routing byte definitions.

RM8 System Function Network

This section contains diagrams of the PS 390 system function network. The diagrams show the logical paths of the routing bytes and functions.

RM9 Initial Structures

This section describes initial data structures created at power-up. Configure mode is discussed and a runtime system is defined.

RM10 Terminal Emulator

This section gives instructions for changing the modes and features of the terminal emulator by either sending escape sequences from the host, entering PS 390 commands in the SITE.DAT file, or sending the appropriate ASCII characters to terminal emulator functions.

RM11 System Errors

This section is a compendium of all user error messages (informational, recoverable, fatal, and warning). Error messages are listed in numerical order. The text of the message is given with an indication of common causes of the error and, where appropriate, ways to correct it.

RM12 Diagnostic Utilities

This section provides a reference for the utility commands that are on the PS 390 diagnostic utility diskette.

RM13 Interactive Devices

This section describes how the PS 390 interactive devices work and are connected to the system. Interactive devices include a peripheral multiplexer, keyboard, data tablet, function buttons, control dials and mouse.

RM14 GSR Internals

This section describes the data formats expected by the PS 390 command interpreter. It is provided for advanced programmers to write their own GSRs.

RM15 Release Notes

A divider is provided for information supplied with future releases of software.

RM16 Index

This section contains an index to the complete *PS 390 Document Set*.

RM5. HOST COMMUNICATIONS

CONTENTS

TABLE

Section RM5
Host Communications

The PS 390 communicates with a variety of host computers by way of communications interfaces. The standard PS 390 interface is the RS-232-C asynchronous serial communication protocol. Also supported are the Ethernet, Parallel, IBM 3278, and IBM 5080 interfaces.

This section describes the data flow between the PS 390 and the host processor. The initial sections introduce some of the basic concepts of data communication, particularly those directly affecting the interface to be set up between the PS 390 graphics system and the host computer.

1. Host/PS 390 Interface

One of the most important considerations in setting up the configuration characteristics of a PS 390 graphics system is the interface between the host computer system and the PS 390.

The standard data communication interface to the PS 390 is an RS-232-C asynchronous serial line. The terms "asynchronous" and "serial" refer to two important communication characteristics.

Binary data may be transferred between electronic devices in "serial", over a single line, or in "parallel", over several lines at once, by changes in current or voltage. In serial transmission, the bits that represent a character are sent down a single wire, one after the other. These serial signals are converted to parallel form at the reception end by shift registers. (In most data communications applications, serial transmission is preferable to parallel transmission, since fewer wires must be run. However, parallel transmission is faster, as more data can be sent across the line at once.)

Data transfers may be of a "synchronous" nature, where the exact bit framing of each byte of information is coordinated for the entire message by the transmission of two or more synchronization characters at the beginning of the message. All characters that follow these characters occur within a specific time frame called a "character time."

Or, data transfers may be of an "asynchronous" nature, where each character is self-defined by the use of a start bit and one or more stop bits. The start and stop bits occur before and after the byte of data. For this reason, this mode of transmission is referred to as "Start/Stop Transmission." In this mode, the arrival time of each character is random. Each end of the transmission line must know what the transmission rate is to sample the line at correct intervals following the receipt of a start bit.

Under PS 390 graphic system protocol, the RS-232-C standard interface sends data signals over a single, serial line using asynchronous transmission. The PS 390 may also be interfaced to a DEC/PDP11 or a DEC/VAX host over an asynchronous parallel line.

RS-232-C refers to a standard for interface communication set by the Electronic Industries Association (EIA). The RS-232-C standard contains:

- The electrical signal characteristics.
- The interface mechanical characteristics.
- A functional description of the interchange circuits.
- A list of standard subsets of specific interchange circuits for specific groups of communication system applications.

It is important when reviewing specifications for computer/system interfaces to understand what the various interface leads do, and which are essential for proper interface between the PS 390 graphics system and the host computer.

1.1 RS-232-C Specifications

The physical connection between the PS 390 and the host is made through plug-in, 25-pin connectors (Cannon or Cinch DB Series). These connectors are keyed for 13 pins on the top row, and 12 pins on the bottom row. The PS 390 ports on the communication connector panel provide the male element for the interface. The pin assignments and signal definitions supported by the PS 390 graphics system are given in Table 5-1.

RS-232-C standard states that the cable between the data communications equipment should be no longer than 50 feet. However, longer cabling distances have been used successfully.

For the PS 390 EIA RS-232-C communication ports, a Control-ON (logical 0), or "SPACE" condition exists if the voltage present is greater than +5 volts and less than +25 volts with respect to signal ground. A Control-OFF (logical 1), or "MARK" condition exists if the voltage present is less than –5 volts and greater than –25 volts with respect to signal ground. This assumes that the PS 390 signal ground and the communication data device signal ground are at the same potential.

Table 5-1. RS-232-C Connector Pin Definitions

PIN #	EIA LABEL	ABBREV. NAME	SIGNAL NAME	DIRECTION
1	AA	GND	Protective ground	N/A
2	BA	TXD	Transmit data	To DCE*
3	BB	RXD	Receive data	From DCE
4	CA	RTS	Request to send	To DCE
5	CB	CTS	Clear to send	From DCE
6	CC	DSR	Data set ready	From DCE
7	AB	GND	Signal ground	N/A
8	CF	DCD	Data carrier detect	From DCE
15	DB	TXCA	Transmit clock	From DCE
17	DD	RXC	Receive clock	From DCE
20	CD	DTR	Data terminal ready	To DCE
24	DA	TXCB	External transmit clock	To DCE

* DCE = Data Communication Equipment

1.1.1 Signal Definitions

The following are definitions of the RS-232-C signals shown in Table 5-1.

- AA, AB (Protective Ground and Signal Ground) — These two grounds are electrically independent. Protective Ground connects to the power ground. Signal Ground connects to the logic ground. No direct frame grounding occurs at the connector. Strict EIA RS-232-C standard definitions are not directly applicable.

- BA (Transmit Data) — Data from the PS 390 are transmitted on this line. The signal is generated by the PS 390 processor.

- BB (Receive Data) — Data are sent to the PS 390 on this line. The signal is passed to the PS 390 via the data communications equipment.

- CA (Request to Send) — This signal is generated by the PS 390 processor. The output may be programmed to conform with EIA RS-232-C protocol. Generally, an "ON" CA (request to send) signal indicates the PS 390 processor is ready to transmit information.

- CB (Clear to Send) — This signal may be generated by data communication equipment. An OFF condition will terminate data transmission. An ON condition allows data transmission to resume. If no connection is made, an internal pull-up resistor will assert this line to an ON condition (+12V) for non-standard RS-232-C communication.

- CC (Data Set Ready) — This signal may be generated by the data communication equipment. The function of this signal is controlled by software within the PS 390 processor. Usually, an 'ON' CC (data set ready) is sent by the data communication equipment to indicate that it is ready to transmit.

- CF (Data Carrier Detect) — This signal may be generated by the data communication equipment. ON assertion of this signal allows BB (receive data) to be accepted by the PS 390 processor. If no connection is made, this line will be pulled to an ON condition (+12V) to allow non-standard EIA RS-232-C communication. To disable the BB (receive data) communication, an OFF condition must exist. Definition of this pin is software controlled for Port 1 of the PS 390 processor.

- CD (Data Terminal Ready) — This signal is generated by the PS 390 processor and is under software control. When asserted to an ON level, CD indicates that the PS 390 processor is ready to communicate.
- DA TXCB (Transmit Clock B) — This signal is generated by the PS 390 processor. DA provides a timing clock to indicate the center of each element of data. This timing clock can either be equal to the transmitted data frequency, or equal to 16 times the data frequency. DA TXCB is under software control. Port 1 of the PS 390 processor does not directly generate this signal. It relies on TXCA (transmit clock A) to generate this clock.
- DB TXCA (Transmit Clock BA) — This input signal is generated by external transmitting data communications equipment. This clocking signal input can control the rate at which the PS 390 processor transmits data out. The ability to use this clock input is software controlled.
- DD RXC (Receive Clock) — This input signal is generated by external transmitting data communications equipment. This clock determines the rate at which the PS 390 processor receives data. The ability to use this clock is software controlled.

1.2 RS-232-C Cabling, Connectors and Pins

All cabling and connectors used in the interface between the PS 390 and the host system must be provided by the user.

A null-modem cable configuration may be necessary to correctly connect the pin signals through the RS-232-C interface.

Cables and the 25-pin connectors for RS-232-C are available through most major computer product supply centers.

The cables running from the host to the PS 390 processor should terminate with a female connector, as the PS 390 data communication ports house male elements.

The decision to use shielded or unshielded cable is left to the user. Shielded cable is highly recommended in noisy environments, but typically it has a higher capacitance per foot than unshielded cable, which may reduce the operating speed.

2. PS 390 Serial Communication Characteristics

This section describes the serial I/O parameters the PS 390 graphics system has defined for each port. The defaults (values assigned to each port when the system is powered on in standard configuration) for the data characteristics are listed in this section. For information on how these values can be configured in a bootable file on the PS 390 graphics firmware diskette, refer to section 2.3. The following information applies to PS 390 graphics systems asynchronous transmission:

- The baud rates available on Ports 1, 3, and 4 on the PS 390 are: 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 9600, and 19200. Port 5 runs at 19200.
- The PS 390 may be configured for 5, 6, 7, or 8 bits per character, although the host port must pass all characters of the 7-bit ASCII character set (for example 7 or 8 bits per character).
- Only one start bit will be accepted (and generated) by the PS 390.
- The PS 390 will accept (and generate) 1 or 2 stop bits.
- The PS 390 and the host can communicate using an XON_XOFF protocol. In this protocol, control sequences are generated that tell the sender (either the PS 390 or the host) when to start (XON), or stop (XOFF) data transmission. These control sequence values default to CTRL S (DEC 17 character) for XON, and CTRL Q (DEC 19 character) for XOFF. Under XON_XOFF, bit stripping is controlled by the /MASK_TO_7 BITS option.

  Additionally, there are available values for data characteristics that are unique to the XON_XOFF protocol. These values and their definitions are shown in section 2.2.
- The PS 390 will run with even, odd, or no parity. Parity is a character checking device that operates by adding non-information bits to data, making the total number of ones in each grouping of bits either odd for odd parity, or even for even parity. This permits error detection for an odd number of incorrect bits in each group.

- Each port may be configured to cause a trap to the PS 390 Debugger in the event a break is detected on that port.
- The PS 390 may be set to hold a maximum number of 127 buffers to hold data transmitted from the host. The default is eight buffers. Each buffer may be set to a maximum of 32,767 bytes, with the default at 48 bytes per buffer. This option allows the user to specify the amount of memory space to be allocated for data reception from the host. The user may specify the number of free input buffers below which the host will be sent an XOFF to suspend transmission. The number of free buffers above which the host will be sent an XON to resume transmission may also be specified.

2.1 Asynchronous Port Defaults

The defaults for Ports 1, 3, 4, and 5 are:

- Port 1 — Host Port - 9,600 baud, 8 bits per character, 1 stop bit, no parity, no_mask_to_7_bits, transparent mode. Sends all XON_XOFF protocol characters, ignores incoming XON_XOFF (no_hear_XON), 8 48-byte buffers with 0 STOP buffers and 1 GO buffer, and debug break disabled.
- Port 3 — Debug Port - 9,600 baud, 8 bits per character, 1 stop bit, no parity, non-transparent mode that accepts all XON_XOFF protocol characters, 8 48-byte buffers with 0 STOP buffers and 1 GO buffer, and debug break enabled.
- Port 4 — 300 baud, 8 bits per character, 1 stop bit, no parity, non-transparent mode that accepts all XON_XOFF protocol characters, 8 48-byte buffers with 0 STOP buffers and 1 GO buffer, and debug break disabled
- Port 5 — Multiplexer Port - 19,200 baud, 8 bits per character, 1 stop bit, no parity, transparent mode that does not recognize XON_XOFF protocol characters, 8 48-byte buffers, and debug break disabled.

The status of all the ports may be verified by using the SHOW INTERFACE command.

2.2 Changing Port Status

The following command sequence can be used to change any of the default values on Ports 1, 3, 4, and 5. These new values must be within the acceptable values for data characteristics as given in the previous section. The port values are changed by entering the command:

```
SETUP INTERFACE <name>/<options>;
```

where name is the port being reconfigured, options refers to the option setting the communications interface. The command:

```
SHOW INTERFACE <name>;
```

where <name> is the port, can be used to check the values of a given port.

In using these commands, the names of the ports are as follows:

Port 1 is designated port10
Port 3 is designated port30
Port 4 is designated port40
Port 5 is designated port50

The available options for SETUP INTERFACE are:

/SPEED=<baud rate> — input and output communications speed between 50 and 19200.

/EVEN_PARITY — establishes monitoring of parity on input and generation of parity on output, using EVEN parity.

/ODD_PARITY — establishes monitoring of parity on input and generation of parity on output, using ODD parity.

/NO_PARITY (default) — terminates the monitoring of parity on input and generation of parity on output.

/BITS_PER_CHARACTER=<number of bits per char> — sets the width of a character in bits (normally 8, including 7-bit ASCII).

/STOP_BITS_PER_CHARACTER=<number of stop bits per char> — sets the number of stop bits for each character (normally 1).

/XON_XOFF — enables the PS 390 to use XON_XOFF protocol to tell the host (or device) on this port to resume or suspend transmission. Default is to this protocol.

/NO_XON_XOFF — disables the use of XON and XOFF protocol from the PS 390 to the host (or device) on this port to resume or suspend transmission.

/HEAR_XON — enables the use of XON_XOFF protocol for the host (or device) on this port to tell the PS 390 to resume or suspend transmission.

/NO_HEAR_XON — disables the use of XON_XOFF protocol for the host (or device) on this port to tell the PS 390 to resume or suspend transmission. Default is NO_HEAR_XON.

/BREAK — enables the receipt of a BREAK on this port to call the ROM debugger.

/NO_BREAK — disables the receipt of a BREAK on this port to call the ROM debugger. Default is NO_BREAK.

/SPEED_EXTERNAL — sets the port speed to that of an attached modem, rather than from an internal clock. (This applies only to those ports with full modem support.)

/NO_SPEED_EXTERNAL — tells this port to use its internal clock, at the speed set by /SPEED=. Default is NO_SPEED EXTERNAL.

/BUFFERS=<number of buffers> — specifies the number of buffers in the input pool. Default is 8 buffers.

/BUFFER_SIZE=<number of bytes> — specifies the size of each buffer in the input pool. Default is 48 bytes.

NOTE

If input is received continuously, buffers will be filled until they are full. The buffer size will, in this case, specify the quantum of input being processed by subsequent functions.

If input is received at much less than the maximum baud rate, buffers will be released to waiting functions after 2 character times without receipt of a byte. In this case, the strict product of <buffer size> and <number of buffers> will not be the true amount of input buffering.

/N_STOP_BUFFERS=<number of buffers> — specifies the number of free input buffers below which the sender is told to suspend transmission. This has no effect unless the port is in /XON_XOFF mode. Default is 1 Stop Buffers. This is for host to PS 390 communication only.

/N_GO_BUFFERS=<number of buffers> — specifies the number of free input buffers above which the sender is told to resume transmission. This has no effect unless the port is in /XON_XOFF mode. Default is 2 Go Buffers.

The following four commands allow the user to specify non-standard X_ON-X_OFF characters:

> /SEND_XON_CHAR=<char code> — specifies the character code as an integer (defaults to decimal 17) to be sent out from the PS 390 to tell the sender to resume transmission. This has no effect unless the port is in /XON_XOFF mode.
>
> /SEND_XOFF_CHAR=<char code> — specifies the character code as an integer (defaults to decimal 19) to be sent out from the PS 390 to tell the sender to suspend transmission. This has no effect unless the port is in /XON_XOFF mode.
>
> /OBEY_XON_CHAR=<char code> — specifies the character code as an integer (defaults to decimal 17) that, when received by the PS 390, allows the PS 390 to transmit.
>
> /OBEY_XOFF_CHAR=<char code> — specifies the character code as an integer (defaults to decimal 19) that, when received by the PS 390, stops the PS 390 from transmitting.

/MASK_TO_7_BITS — specifies that incoming bytes are to have their 8th bit, normally the parity bit, stripped off.

/NO_MASK_TO_7_BITS — (default) specifies that incoming bytes are not to be masked.

/BREAK_TIME=<break time> — specifies the length of time in centiseconds that an outgoing BREAK is to be held. This defaults to 10. Maximum = 127. (Section *IS3* contains instructions for defining the break key.)

/ASYNCHRONOUS — normal mode of operation.

All commands are terminated with a semicolon (;) and a carriage return. The menu available with the SHOW INTERFACE command lists only those parameters that are relevant to the interface.

2.3 Changing PS 390/Host Interface Values Using the SITE.DAT File

Port values may be changed to suit specific site requirements in two ways: the default values can be changed by using the SETUP INTERFACE commands in configuration mode, or the SETUP INTERFACE commands can be entered into the SITE.DAT file. If the value needs to be changed for just one session, so that the port will go back to its default values during the next boot-up, the SETUP INTERFACE command can be entered during a PS 390 session. Should the new port value need to be installed more permanently, with the new value booted instead of the default, the SETUP INTERFACE commands should be entered into the SITE.DAT file.

Any of the SETUP INTERFACE commands can be entered in the SITE.DAT file, using the following forms:

```
SETUP INTERFACE portn/option;

SETUP INTERFACE portn/option=<p>;
```

where **n** is the port name, **/option** is the name of the feature being set, and <p> is the specified parameter.

Examples:

```
SETUP INTERFACE port10/XON_XOFF;
```

would enable Port 1 to use XON_XOFF protocol to tell the host (or device) on this port to resume or suspend transmission.

```
SETUP INTERFACE port10/SPEED=2400/XON_XOFF;
```

would set Port 1 to a baud rate of 2400 and enable XON_XOFF protocol.

3. PS 390 Transmission Protocol and Error Detection

This section details the transmission protocol necessary to receive and transmit data over the asynchronous interface. It also provides a brief description of the three types of errors detected by the Enhanced Programmable Communications Interface (EPCI) status register.

3.1 PS 390 Transmission Protocol

The PS 390 graphics system uses an XON_XOFF handshaking protocol to maintain orderly data communication over a full duplex, asynchronous, serial line between itself and the host computer. The receiver of XOFF (decimal 19) is to suspend transmission as soon as possible. The receiver of XON (decimal 17) is to resume transmission until the next reception of XOFF. The PS 390 will suspend transmission within one character time and can accept up to one buffer full of characters after XOFF is sent.

The following equation shows how many bytes of an empty buffer are left when an XOFF is sent. An XOFF will be sent to the host that many bytes before input buffering is exhausted.

```
((Number of STOP buffers +1) * Number of bytes/buffer) - 1
```

3.1.1 Data Reception and Transmission

The PS 390 defaults to eight 48-byte buffers available to receive data from the host computer. Transmitted characters are placed in the first free buffer starting in the first position and continuing to the end of the buffer. When the buffer is full, the next available buffer is used. If all allocated buffers are full, the PS 390 will drop everything off the line until a buffer is free.

When the XON_XOFF protocol is used, the PS 390 will send an XOFF to the host (sender), when the number of free buffers is equal to the number of STOP buffers. The PS 390 will send XON to the host when the number of free buffers is equal to the number of GO buffers.

An XOFF received on the host input port disables data transmission from the host to the PS 390 until the PS 390 sends an XON. If a host transmission aborts before XON is transmitted, or if the host transmits XOFF as part of the LOGOFF message, it is necessary to manually clear the XOFF condition. XOFF is cleared and the port re-enabled for transmission whenever a SETUP or SHOW INTERFACE command is executed.

Rebooting the PS 390 will also clear the XOFF condition.

Default for the PS 390 is NO_HEAR_XON_XOFF.

3.1.2 Data Transmission Without XON_XOFF

Operation without support of the XON_XOFF protocol is discouraged. If XON_XOFF protocol is not available on the host, it is up to the user to ensure that an adequate number of buffers are allocated for data reception on the PS 390.

3.1.3 Transmission Errors

If the XON_XOFF protocol is not used, and the number of available buffers is not large enough to hold the incoming data from the host (sender), data characters will be lost. These lost characters are detected and counted by the input routines. The SHOW INTERFACE command will give the current error counts for each port.

Messages which characterize lost input characters are:

- PARSER SYNTAX ERROR due to bad syntax generated by the lost characters
- ERROR E 12 *** Message which function cannot handle

3.2 Transmission Error Detection

The Enhanced Programmable Communications Interface (EPCI) used on PS 390 Ports 1, 3, 4, and 5, is able to detect three types of transmission errors. When one of these transmission errors occurs, a bit is set in the EPCI status register where it can be read by the graphics control processor. The errors detected are:

- Parity errors (if parity is enabled)
- Framing errors
- Overrun errors

The SHOW INTERFACE command will display all errors detected from the last PS 390 boot.

3.2.1 Parity Errors

The parity bit follows the character bits in data transmission. If there are 7 bits/characters, and parity is enabled, the total number of bits is 8 with the parity bit being the last transmitted bit. Ignoring the start bit and stop bit(s), the letter "A" when transmitted with EVEN parity would appear as follows:

lsb						msb	
1	2	3	4	5	6	7	parity
1	0	0	0	0	0	1	0

where "lsb" is the least significant bit and "msb" is the most significant bit.

The same character transmitted with ODD parity would look like this:

lsb						msb	
1	2	3	4	5	6	7	parity
1	0	0	0	0	0	1	1

EVEN parity sets the state of the parity bit such that the number of ones in the 8 bits is an even number.

ODD parity sets the state of the parity bit such that the number of ones in the 8 bits is an odd number.

If parity is enabled, the EPCI determines the parity of the received character and compares this parity with the parity bit transmitted. If they do not agree, the parity error flag is set in the EPCI status register.

From the example of the character "A", it can be seen that if the host and the PS 390 do not agree on the parity being used, every character received or transmitted will generate a parity error.

This vertical error detection scheme can only discern an odd number of bit errors. For example, if bits 2 and 3 are erroneously changed to ones, so that the character transmitted appears to be:

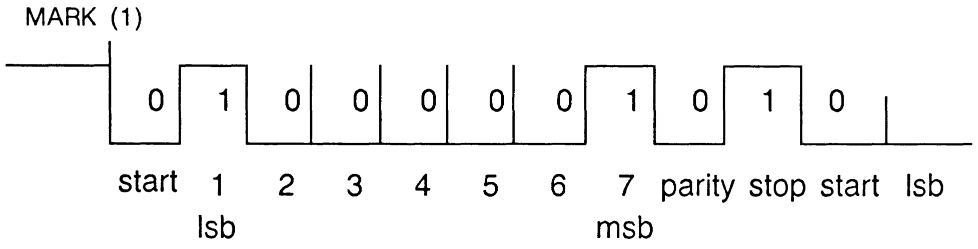
lsb 1	2	3	4	5	6	msb 7	parity
1	1	1	0	0	0	1	0

EVEN parity — the parity bit is correct for the character received ("G") but incorrect for the letter sent ("A").

The PS 390 supports ODD and EVEN parity, or NO parity.

3.2.2 Framing Errors

"Framing" is the process of determining which group of bits constitute a character. An error in this process is called a "framing error". Characters are framed by the start bit and the stop bit(s). Looking at the character "A" again (assume one stop bit):



The line is held in a MARK condition with current flowing when characters are not being transmitted. If for some reason the EPCI failed to detect the start bit when the signal goes to an ON, or SPACE condition, it is possible that it would assume bit 2 was the start bit, and bit 3 was the lsb, etc. At the time EPCI expected to see a stop bit, it would instead see the lsb of the next character, and a framing error would occur. When a framing error does occur, the EPCI sets the framing-error flag in the status register.

3.2.3 Overrun Errors

An overrun error occurs when the JCP fails to read the characters in the holding register of the EPCI before the next character received is placed in the holding register. When this happens, the EPCI will overwrite the contents of the holding register with the next character. This overwrite causes the overrun error flag to be set in the EPCI status register.

4. Methods of Communication over the Host Line

Section *IS3* discusses the various methods of data communication that can be used over the PS 390/host line. These methods include standard ASCII transmission or the GSRs, an E&S supplied host-resident software package.

The GSRs perform all prepackaging of data prior to sending it in binary format to the PS 390. The routing bytes required to channel the data to the proper PS 390 system function are contained within the routines. The routines build data 'packets' that include all the necessary information to process the data, and are in a form that is immediately acceptable by the PS 390 system function, F:CIROUTE.

In all cases, F:CIROUTE expects to receive data in a specific format called packets. These packets may be in either ASCII or binary, and for asynchronous communication, may be in either count or escape mode. Over the parallel interface, these packets are sent only in count mode.

When communicating with standard ASCII transmission, the PS 390 system functions (data reception functions, such as F:DEPACKET) that interface between the system and the hardware are responsible for building the data packets. The routing bytes that are used to channel data to the appropriate PS 390 system function must be supplied. A brief description of the routing bytes and their channels can be found in Section *RM7*.

The following sections deal with the use of count and escape mode in asynchronous data transmission.

4.1 Data Communications — Escape and Count Mode

Data is sent to the PS 390 from the host as a stream of bytes. These bytes must contain information that is intelligible to PS 390 system functions about the nature of the message and where it is to be sent internally in the PS 390. The descriptions that follow describe the data transfer modes used in host/PS 390 communication and briefly describe the system functions that accept, examine, and route data internally in the PS 390.

Data may be transported over an asynchronous line in two modes: escape mode or count mode. The mode used is dependent on the application and can be selected by the user. Count mode is the faster mode, as the system function, F:DEPACKET, that converts a stream of bytes into a stream of packets does not have to check the identity of each byte.

A system function, F:DEPACKET, accepts data input to the PS 390 from the host. F:DEPACKET converts a stream of bytes from the host into a stream of Qpacket/Qmorepacket. A Qpacket is a block of character data that can be sent from one PS 390 function to another. When data comes from the host through the F:DEPACKET function, it contains a byte for routing control. A Qmorepacket is a Qpacket that when coming from the host through F:DEPACKET, has no routing byte (i.e. a Qmorepacket has the same destination as the previous Qpacket.)

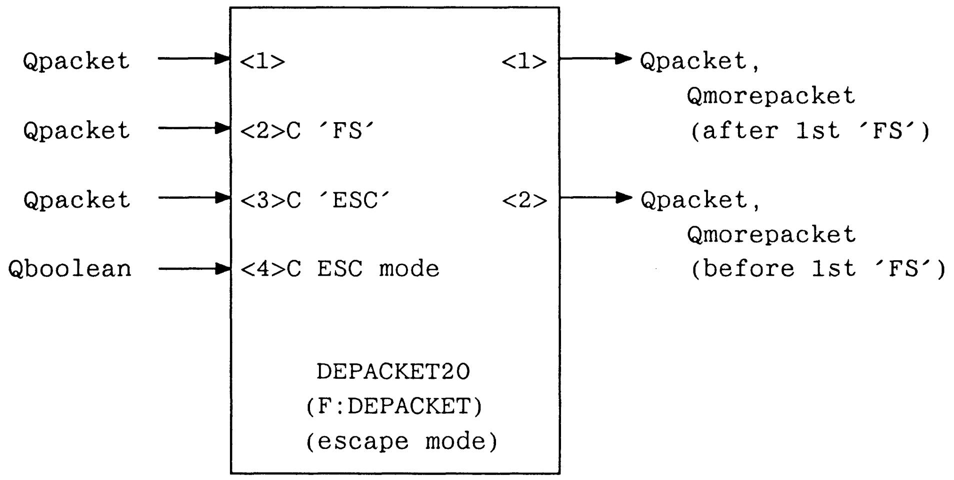
There are two instances of the F:DEPACKET function. The first, DEPACKET0, accepts all incoming bytes from the host on input <1>. It channels all incoming data through to output<2> until it sees the Start of Packet (SOP) character <ACK> (ACKNOWLEDGE — decimal character code 06, ASCII ↑F) that signifies the start of a count mode packet.

All the data sent through to output<2> of DEPACKET0 are sent to input<1> of the second DEPACKET function, DEPACKET20, which then checks all incoming data for the SOP character <FS> (Field Separator — decimal character code 28, ASCII ↑\) that signifies the start of an escape mode packet. It will also route all incoming bytes out output<2> until it sees the <FS> character. Output <2> of DEPACKET20 is connected to ES_TE1 (the screen).

These instances of F:DEPACKET are described below. The characters that are used to signify SOP (<FS> and <ESC> characters) may be changed by the user by sending the new characters to the correct inputs of F:DEPACKET.

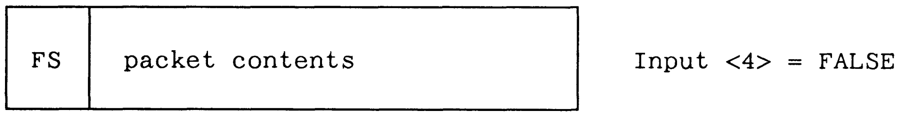
4.1.1 Escape Mode

In escape mode, F:DEPACKET looks at every byte to see if it is a SOP character, which by default in escape mode is the ASCII Field Separator <FS> character, or an <ESC> character.

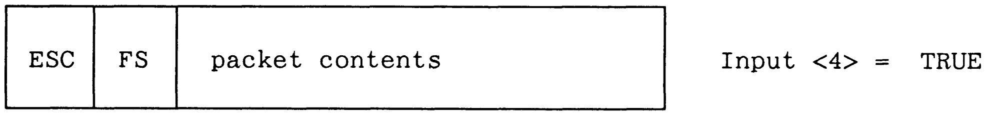


Qpacket
Qpacket
Qpacket
Qboolean
<1>
<2>C 'FS'
<3>C 'ESC'
<4>C ESC mode
<1>
<2>
Qpacket, Qmorepacket (after 1st 'FS')
Qpacket, Qmorepacket (before 1st 'FS')
DEPACKET20 (F:DEPACKET) (escape mode)

In escape mode, F:DEPACKET assumes that a packet is defined as either:



or



where <FS> represents the SOP character that is by default the decimal character code 28 (↑\).

The definition of FS (one character) is taken from a single character Qpacket on input <2>.

In the first mode (input <4> = FALSE), any FS or ESC characters within the message packet must be escaped by prefixing them with an ESC character (i.e. the <ESC> character, decimal character code 16 (↑P)). Thus <ESC><x> becomes <x> for all values of x.

In the second mode (input <4> = TRUE), only ESC characters within the message packet must be escaped by prefixing them with an ESC character.

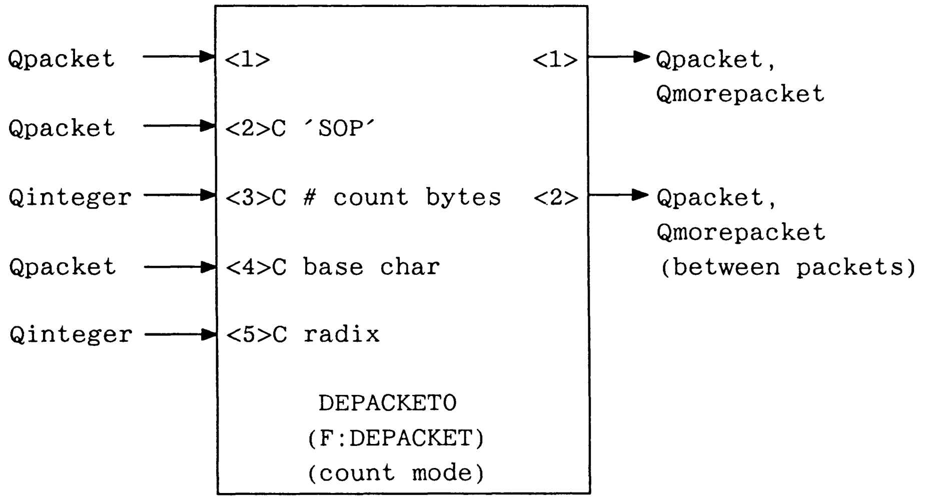
The ESC character is defined by a single character Qpacket on Input <3>. Output <1> outputs Qpacket and Qmorepackets of any messages after the first SOP control character is received. Output <2> outputs Qpackets and Qmorepackets of any messages before the first SOP control character is received. A Qpacket is output on Output <1> each time a SOP control character is received. Otherwise Qmorepackets are output.

Output <2> is normally connected the Terminal Emulator Input and Output <1> is connected to F:CIROUTE for both Count and Escape Modes.

The routing path will be used for data transfer until the multiplexing function sees another SOP character, and a packet with another routing byte.

4.1.2 Count Mode

In count mode, once the SOP <ACK> character is seen, F:DEPACKET merely counts the bytes until the count is reached. No attempt is made to decode any bytes until the count is reached. Because F:DEPACKET does not examine the data, it is faster than escape mode, where all bytes are checked by F:DEPACKET to see if they are <FS> or <ESC> characters. Also, count mode allows for the use of any <SOP> or <ESC> sequences as part of the data.



In count mode, F:DEPACKET assumes that a packet is defined as:

SOP	count bytes	packet contents

where SOP represents the Start of Packet character that is by default the the ASCII <ACK> character, decimal character code 06 (↑F).

The definition of SOP (one character) is taken from a single character Qpacket on input <2>.

The message count is defined by n bytes (n defined by the Qinteger on input <3>). Each count byte is offset from the base character (the base character is taken from a single character Qpacket on input <4>). After the base character is subtracted, each count byte becomes a digit of the message count whose radix is defined by the Qinteger on input <5>.

Output <1> outputs Qpackets and Qmorepackets of count mode messages. Output <2> outputs Qpackets and Qmorepackets of any messages which are not in count mode.

The <SOP> byte and the count bytes are removed from the start of the packet before the packet is sent to F:CIROUTE, which performs the actual routing.

4.2 Using the Routing Bytes for Local Data Flow

For asynchronous interfaces, routing can be done in a number of different ways; but every data transfer must be preceded by an <ACK> character (count mode) or an <FS> character (escape mode), and a routing byte that gives the destination of the data. If ASCII data are to be sent from the host to the Command Interpreter (in the Escape Mode), the file containing the Command Interpreter routing bytes must precede the data, and must contain the following characters:

↑\0 where ↑\ is a CTRL backslash

To route the line from the Command Interpreter back to the Terminal Emulator, a file should contain the following sequence:

↑\>

Routing back to the Terminal Emulator is essential if the Terminal Emulator is being used to download the file. To get the host prompt back after downloading the file, the line must be routed back to the Terminal Emulator mode (↑>). If the routing byte was not sent, the following command can be entered from the keyboard in command mode to route back to the Terminal Emulator:

```
SEND TRUE TO <1>RESET_TE;
```

If the Escape Mode <FS> characters appear as data in the PS 390 command file, they must be prefixed by the escape sequence DLE (↑P). The ↑P (decimal 16), when immediately preceding the FS characters, will identify the characters as being non-muxing data to be passed along.

The ↑\ <FS> character, the ↑F <ACK> character, and the escape sequence (↑P) can be changed by the user in the SITE.DAT file. This should be done when the sequences used with the PS 390 are incompatible with the host or have another site-specific value.

4.3 Changing the <ESC>, And/Or <SOP> Sequence Characters in the SITE.DAT File

If the <ESC>, and/or <SOP> sequence characters used by E&S are incompatible with the host, or have another site-specific value, these characters can be changed by sending new values for these sequences to an instance of F:DEPACKET in the PS 390.

These new values must be included as PS 390 commands in the SITE.DAT file that is loaded during the system power-up. These commands should never be sent down from the host or entered in from the PS 390 keyboard during host transmission.

NOTE

> If the <ESC> or <SOP> characters are changed in the SITE.DAT file, this change must be incorporated in the GSRs, as these routines use the same sequences for routing.

The PS 390 command for changing the escape mode <SOP> (default is <FS>, decimal character code 28, ASCII character '↑\') character is as follows:

```
    SEND CHAR(I) to <2>DEPACKET20;
```

where I is the integer value corresponding to the new <SOP> character in escape mode.

The PS 390 command for changing the escape mode <ESC> character is as follows:

```
    SEND CHAR(I) TO <3>DEPACKET20;
```

where I is the integer value corresponding to the new <ESC> sequence.

The count mode SOP character, (ASCII <ACK>, decimal character code 06, ASCII ↑F), can be changed by sending the new integer value to <2>DEPACKET0:

```
    SEND CHAR(I) TO <2>DEPACKETO;
```

5. PS 390/IBM Host Communications

The following sections describe the data flow between the PS 390 and IBM host processors. An introduction to the basic concepts of data communication, particularly those directly affecting the 3278 interface, are discussed first.

5.1 PS 390 Data Communication

It is intended that all communication between the IBM host and the PS 390 use the cross-compatibility software provided to the user as the Graphics Support Routines (GSRs). The GSRs reside on the host as either FORTRAN subroutines or Pascal procedures, and are provided to support the interface between the IBM 3274 Controller and the PS 390 Graphics System. The PS 390 is an ASCII system, expecting and generating ASCII characters. The IBM 3274 Controller is an EBCDIC system and is unable to generate the ASCII characters expected by the PS 390. The GSRs provide an interface that allows the two systems to respond to each other. Data that affect messages and message routing internally in the PS 390 are embedded with the software communication routines and are, for the most part, transparent to the user.

5.2 Data Destinations

Data going from the host to the PS 390 have two possible destinations: the PS 390 Command Interpreter (CI) or the PS 390 Terminal Emulator (TE). Data for the CI can be initiated with a GSR or specific ASCII commands.

There are several PS 390 system functions that pass and route data through the PS 390, prior to the command interpreter. These functions, and the data paths, are discussed in section 5.5 and in Section *RM7*. The format of data expected by the CI is given in Section *RM14*.

5.3 Write Structured Field

Graphics data intended for the CI are sent from the host to the PS 390 using a special 3278 command called Write Structured Field (WSF). The WSF command is normally used by the IBM 3274 Controller to create non-keyboard type symbols for use in business graphics applications. All non-WSF commands cause the terminal emulator to perform like a 3278, but Evans & Sutherland has reserved the use of the WSF command to transfer graphics data, because the Load Program Symbols option of the WSF command allows binary data to be sent unchanged to the PS 390. The use of the WSF command requires the 3274 to have support for Programmed Symbols, an option of Configuration Support C, in the 3274 Control Unit. When the GSRs are used, the PS 390 will appear to the graphics application exactly as it would in any other environment. The communication routines of the software will insert the user data in a WSF buffer, and perform all necessary data transfers with the 3278 Terminal Emulator.

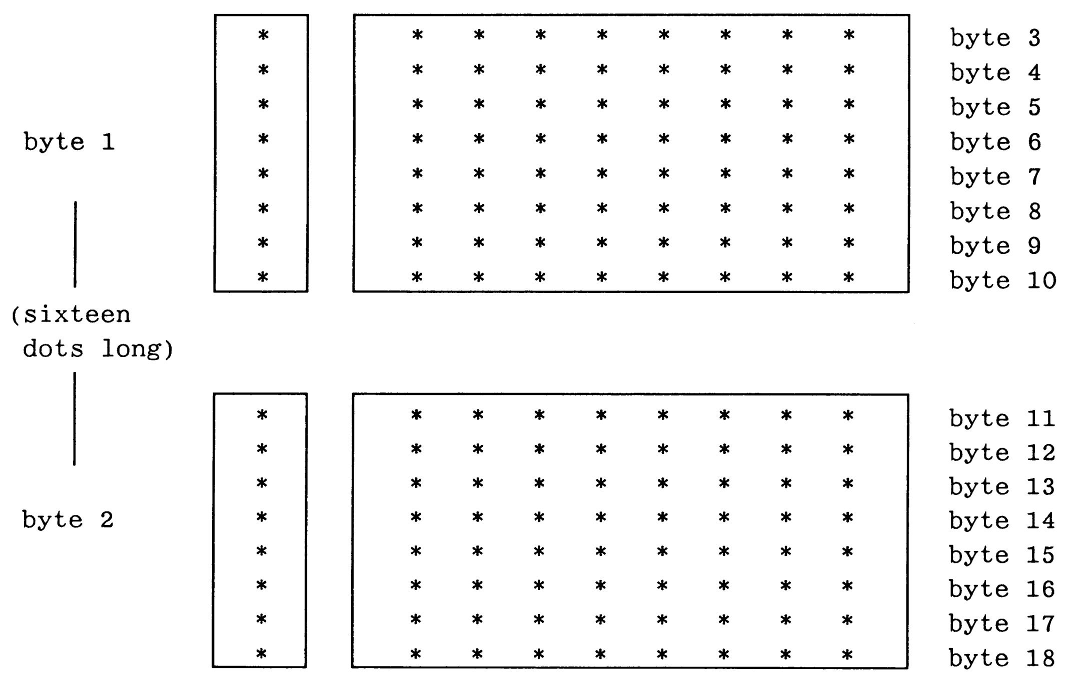
If the GSRs are not used, the user will need to have some understanding of how Programmed Symbols work and how the 3274 sends the symbols to the 3278 to understand how the WSF data buffers are built. A detailed description of Programmed Symbols and their use to transfer graphics data is provided below.

5.3.1 Programmed Symbols

Each symbol displayed on the 3278 screen is composed of illuminated dots made from a nine-by-sixteen dot matrix. The Load Program Symbols function of the WSF command allows users to specifically illuminate any particular set of dots in the matrix to create their own special symbol by setting the corresponding bit in the matrix description to a one. The matrix is described by overlaying it with a set of eighteen eight-bit bytes (9x16=8x18=144).

The following diagram shows how each character matrix is overlaid with eighteen bytes.

CHARACTER MATRIX
(nine dots wide)

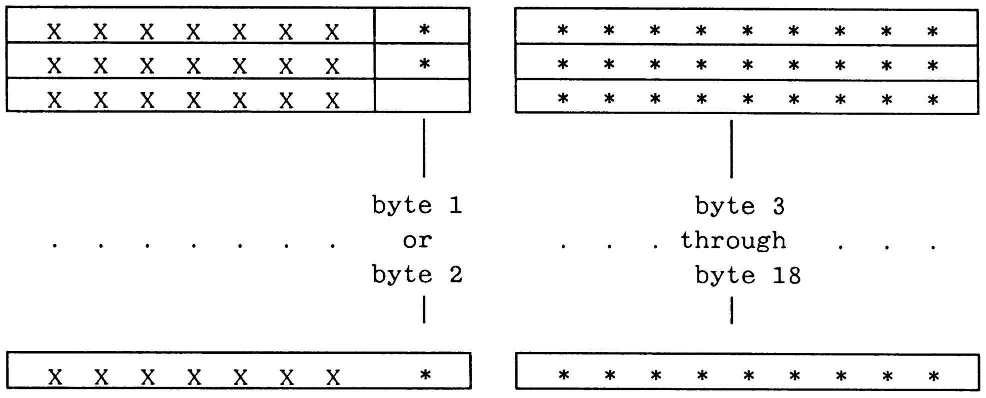


The data that describe the matrix are placed in a WSF buffer in the following order.

```
  byte 1    byte 2    byte 3   . . . . . . . . . . .  byte 18
|********|********|********| . . . . . . . . . . . |********|
```

When the 3274 gets the matrix that was sent in the data stream described immediately above, it converts the data back to a format that looks more like the original matrix. The data are sent in sixteen groups of two bytes each. The first seven bits of the first byte are unused, and the last bit of the first byte is from byte 1 or 2 of the bytes sent. The second byte is made directly from bytes three through eighteen.

Data sent from 3274 to the 3278



The PS 390 receives graphics data passed to it from the 3274 in the format shown above. In order for the PS 390 to avoid the difficulty of reassembling the bytes received, it simply discards the first byte of each of the sixteen two-byte pairs for each programmed symbol. This means that the first two bytes in each programmed symbol sent to the PS 390 cannot be used to contain data.

The graphics data are placed in each program symbol matrix in the following manner:

```
|<---unused------>|<-------- graphics data -------->|
  byte 1   byte 2   byte 3  . . . . . . . .  byte 18
|XXXXXXXX|XXXXXXXX|********|. . . . . . . .|********|
```

The 3274 expects the WSF buffer to contain one or more complete program symbols. If the PS 390 graphics data does not fill a complete symbol, the full eighteen bytes of the symbol must be sent, but the remainder is ignored. To know exactly how much graphics data is present, the first two bytes of the graphics data should contain the length of the actual data following. The length does not include the length itself, the first two unused bytes in each program symbol, or any unused bytes following the data in the last program symbol. The length is used only by the 3278 Terminal Emulator, and is external to the graphics data and any multiplexing scheme that may be employed.

The following diagrams show the way the data would be placed in programmed symbols in the WSF buffer.

```
                          length
bytes     |  unused    |  of data   |<--- data ---->|  unused   |<-- data
in each ->|  1     2   |  3  |  4   | 5 |  ... | 18 |  1  |  2  | 3  | ...
symbol

          end of data --->|  unused  |<--extra at end-->|
                   ...| 16|  17 | 18 | 1  | 2  | ...| 18|
```

Note that an extra program symbol was added at the end of the buffer. It is required by the PS 390 to verify that the previous symbol (the last symbol containing data) was received correctly. Note also, that the data did not completely fill the last symbol containing data, but that the full symbol was built.

5.3.2 Load Programmed Symbols

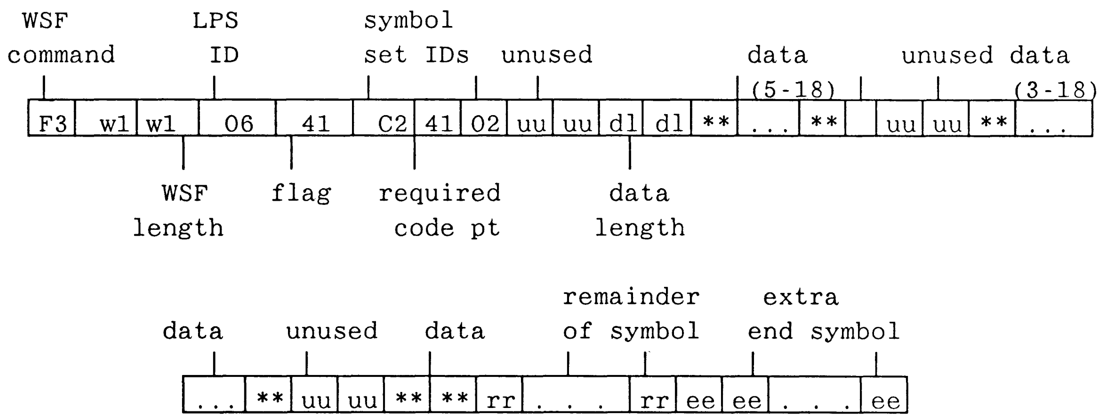
The Load Programmed Symbols option of the WSF command that is used to load the symbols described in preceding paragraphs is invoked by inserting control information after the WSF command code and before the programmed symbols.

The control information contains the following data:

1. A length that includes itself, the control information and all symbols, including the extra one at the end.
2. An identifier that indicates that this is a Load Programmed Symbol request.
3. A flag byte that specifies which options are used.
4. Fields that identify the symbol set that the symbols would be loaded into if this were an actual 3278. This information is not used by the PS 390 and can be any legal value.
5. A starting code point identifier. This value would ordinarily be used to match data from the host to the specific symbol the user wants displayed. The PS 390 uses this value to indicate that the following symbol will contain the data length in its first data bytes and that the first data byte will be a code indicating which output port of the function F:CIROUTE the data will be sent from. A value of X'41' must be used.

The control information can be a constant that is inserted in the buffer, with the length updated to specify the total programmed symbols length.

The final buffer might look like this:



5.4 Configuration of the 3274 Control Unit

To support the transfer of graphics data to the PS 390 using the Write Structured Field command with the Programmed Symbols option, the 3274 Control Unit that supports the interface to the PS 390 must have the Configuration Support C option. Also, the 3274 Control Unit must be customized with the following options:

162 — Structured Field and Attribute Processing (SFAP)

163 — Extended Character Set Adapter

The PS 390 should be included in the total number of devices that require SFAP. Note that this number is a maximum. When the 3274 is initialized, special control blocks needed for SFAP are allocated as needed on a port by port basis beginning with Port 0 until this maximum is reached. SFAP devices attached to subsequent ports will be unable to use the SFAP features until the control unit is re-customized.

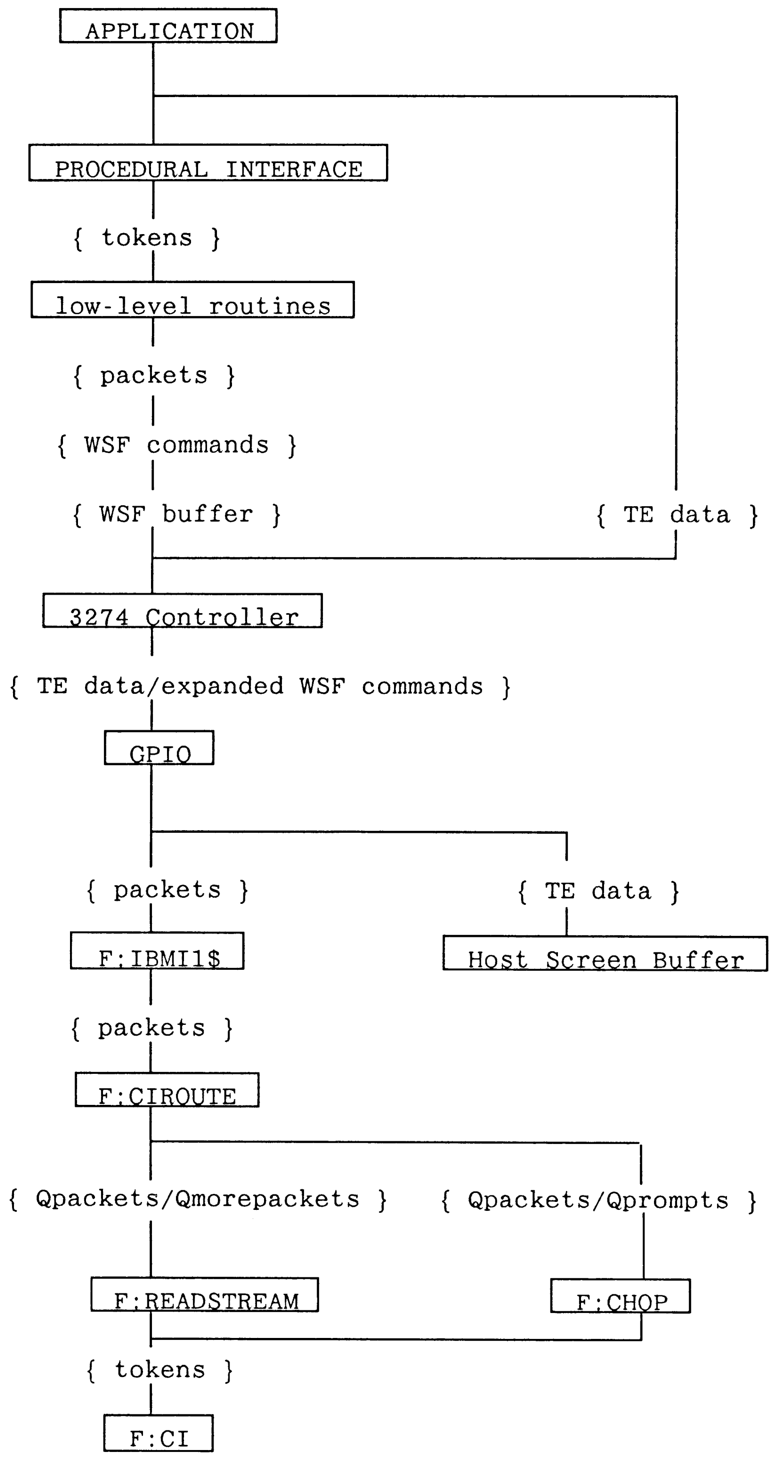
164 — Programmed Symbols

Refer to the appropriate IBM documentation for detailed instructions on the 3274 customization procedure.

5.5 Data Flow Overview

The following diagram illustrates data flow between an application program residing on the host system and the PS 390 system function that initiates graphics commands. In the diagram, routines or functions that pass and/or route data are enclosed on four sides. The format that data are passed in is shown in curly braces.

PS 390/IBM 3278 Interface Data Flow Diagram



There are low-level communication routines supporting the GSRs that use formatting routines to package data for transportation. These routines build WSF envelopes and put the data in outbound PS 390 buffers.

The CI expects "tokens" that consist of a size, a data type, and a value. For a given PS 390 command, the type of command is implicit in the type of one of the tokens. The CI accepts a stream of tokens until it has enough to carry out the command. The GSRs can be thought of as "mailing" these tokens to the CI. The tokens are deposited into several layers or "Qpackets" and "Qmorepackets" of nested envelopes for transportation purposes, but when they reach the CI, they are almost identical to what was built by the GSRs.

A WSF command contains the tokens that are to be sent to the CI. Routing information is included at the head of the WSF command. In the standard PS 390 system, the PS 390 General Purpose Interface Option (GPIO) card takes the routing information and the first 238 bytes of data in a WSF command and puts them into a Qpacket. All subsequent bytes of data in that WSF command are put into Qmorepackets, signifying that the same routing information is to be used. Whenever a WSF command is filled to capacity, or a routing change is required, the current WSF is terminated and a new WSF command is started by the low-level routines. The IBM system I/O services maintain a WSF buffer. The size of this buffer is configurable but generally defaults to a value specified by the routines sending the data. More than one WSF command can go into the buffer and the buffer may be split into smaller pieces when it is sent by the communications access method.

All data bound for the CI are packaged in WSF envelopes. Upon receiving information from the host, the GPIO is able to differentiate graphical data from TE data by the WSF command; anything not in a WSF command is TE data and goes directly to the (Host) Screen Buffer.

Data intended for the CI are passed through a PS 390 routing function, F:CIROUTE. This function expects routing characters at the start of each Qpacket it receives.

The software on the host processor uses routing bytes that will channel the data to the proper PS 390 system function. The routines build the data packets with the routing data embedded in the WSF envelopes. The GPIO repacks these data and passes them, along with the routing information, to the PS 390 system function, F:CIROUTE.

In all cases, F:CIROUTE expects to receive data in a specific format called Qpackets. This function, and an overview of local data flow in the PS 390 is discussed in Section *RM7*.

5.5.1 Modification of Pool Sizes

The PS 390 function SETUPIBM allows the number of empty packets in the input pool for the PS 390/IBM interface system to be modified. The function has one input queue and no output queues. The input queue accepts integer values. At system configuration, the pool size is specified as 256. An example of PS 390 commands used to change the pool size for the IBM system is:

```
SEND FIX(64) TO <1>IBMSETUP1;

SEND FIX(99) TO <1>IBMSETUP3;
```

CUSTOMER INSTALLATION AND USER MANUAL

PS 300 ETHERNET™ INTERFACE

For VAX/VMS Operating Systems

EVANS & SUTHERLAND

March 1987
E&S #901194-073 A1

CHANGED PAGES

This manual has been updated to make it consistent with current firmware capabilities. Pages that have been changed are indicated with an A1 revision level at the top of the page. Because additions to Chapter 6 were fairly extensive, the entire section shows the new revision level.

Chapter 3 and **Appendix A** have notes to A2.V02 users with JCP systems concerning the inclusion of the network node address in the SITE.DAT file.

Chapter 6 has new information regarding Option Bit 3 in the PS Multiplex Message. **Chapter 6** also describes two new functions, Frame Count Request and Frame Count Reply.

PREFACE

This manual contains two levels of customer information: customer installation requirements and user information specific to the interface.

Part I of this manual lists the steps that you, as the customer, must take prior to requesting the hardware installation of the PS 300 Ethernet™ interface. Installation of E&S communication hardware is the responsibility of an Evans & Sutherland customer engineer. E&S hardware installation information given in this manual is for reference only and should not become the basis for unauthorized installation.

Customer requirements include purchase and installation of host-to-PS 300 cabling, the Ethernet host controller circuit board, coaxial cable, transceivers and cables, and repeaters, if necessary. You are also responsible for the installation of the host-resident software (provided by E&S) on the host system.

It is assumed that you have a working knowledge of Ethernet physical and logical layer specifications, Ethernet design and operation, DECnet network software protocol, the VMS operating system, VMS file structures, and VMS commands and utilities.

The first page of Part I of this manual is the Customer Installation Checklist. You must complete this Checklist before requesting the hardware installation of the interface.

Chapter 1 is an introduction to the PS 300 Ethernet interface and protocols. This chapter lists the customer-supplied hardware and software components that must be at the site and installed prior to requesting installation of the interface hardware.

Chapter 2 provides a general description of the VMS/PS 300 software. It is intended for PS 300 customers using the VAX/VMS operating system. You must install the software before requesting hardware installation.

Chapter 3 provides instructions for assigning the PS 300 node address in the host computer and for creating a SITE.DAT file on the PS 300 Ethernet firmware. The SITE.DAT file must be in place before the host can recognize and communicate with the PS 300.

Chapter 4 provides support information on the Ethernet interface, specifically DECnet network protocol. This information is provided to ensure that you are familiar with the PS 300 GPIO - Ethernet network protocol.

Part II of this manual contains advanced user information for the PS 300 Ethernet interface. The **PS 300 Document Set** contains all the user information for the PS 300 systems. The information provided in Part II of this manual is supplemental.

Chapter 5 describes the PS 300 display data structures. Information in this chapter supports the physical I/O capabilities of the Ethernet interface. This information is provided for the advanced PS 300 programmer.

Chapter 6 describes the blocks of data that are used by the Ethernet GPIO to transfer data between the host and locations in the PS 300, such as runtime functions and mass memory. This information is provided for users who wish to write their own host software to communicate with the PS 300 Ethernet interface.

RELATED DOCUMENTS

Information related to the contents to this manual appears in the following manuals:

PS 300 Document Set

The PS 300 Document Set contains system installation, operation, programming, and system-management information. The information is organized into five volumes. Volume 3B of this set describes Pascal V2 and FORTRAN-77 Graphics Support Routines under the VAX/VMS operating system.

PS 350 User's Manual (E&S #901172-092)

This manual provides information specific to the operation of the PS 350 and notes the differences, both hardware and software, between the PS 330 and the PS 350. This document is supplemental to the **PS 300 Document Set.**

The following documents are not supplied by E&S but should be available at your site for reference.

VMS Manual Set

Guide to Networking on VAX/VMS

The Ethernet: A Local Area Network; Data Link Layer and Physical Layer Specifications, Version 2.0., November 1982.

Xerox Corporation
Network Systems Administrative Office
3333 Coyote Hill Road
Palo Alto, California 94304

CONTENTS

PART I

INSTALLATION INFORMATION

PART II

USER INFORMATION

FIGURES

PART I

INSTALLATION INFORMATION

CUSTOMER INSTALLATION CHECKLIST

PS 300 ETHERNET INTERFACE

VAX/VMS

Before requesting hardware installation and testing of the PS 300 Ethernet interface, you, as the customer, must meet the requirements for PS 300 Ethernet interface installation. You have met the requirements when you can answer yes to the questions listed below. Do not request hardware installation until this list is complete.

1. Are you running DECnet on the DEC® VAX/VMS®, version 4.0?
2. Is your VAX communicating successfully using the DECnet network?
3. Is the Ethernet transceiver and transceiver cable to support the PS 300 installed?
4. Is the total length of your coaxial cable less than 500 meters?
5. Is the length of the transceiver cable which will link the PS 300 to the Ethernet coaxial cable less than 50 meters?
6. Are all transceivers on the cable at least 2.5 meters apart?
7. If this is a field upgrade, are all the card levels in your PS 300 at the necessary revision levels? If you do not know the level of your cards, contact the E&S Customer Service Center at 1-800-582-4375.
8. Has your PS 300 node address been installed on your host?
9. Have you built a SITE.DAT file on your host system with the PS 300 address, or are you ready to install the SITE.DAT from the PS 300 keyboard after installation is complete?
10. Have you made sure that a system manager or someone with system privileges will be available during the hardware installation process?

If you have answered yes to all the questions, call E&S Customer Service Center at 1-800-582-4375 and request installation of the interface.

1. INSTALLATION REQUIREMENTS

This chapter should be read carefully before beginning any installation procedures. As the customer, you must meet several requirements before the hardware installation of the interface by an Evans & Sutherland customer engineer can take place. Please follow the procedures outlined in this manual before requesting installation of the hardware at your site.

If there are any questions regarding installation procedures or requirements, please call the Software Support Hotline (800) 582-4375.

INTRODUCTION TO THE INTERFACE

The PS 300 Ethernet interface allows a PS 300 to link to an Ethernet data communications network. Ethernet falls in a middle ground between long-distance, low-speed networks that carry data for hundreds or thousands of kilometers, and specialized, very high-speed connections that are generally limited to tens of meters. Ethernet transmits bursts of message packets at a speed of ten megabits per second. It is intended for use in office automation and distributed data processing environments to allow a selected group of computers to communicate with each other.

All computers on the same Ethernet network are physically connected to a coaxial cable, requiring that all computers be within a relatively close proximity. The coaxial cable is the medium over which the computers in the network communicate.

Each computer in the network has a unique address that distinguishes it from the other computers in the network. Each message (data packet) sent via the coaxial cable includes the address of the computer for which it is intended. Computers in the same network ignore messages that do not contain their particular address.

There are several hardware and software layers that make up Ethernet to allow communications via the coaxial cable.

Basically, all computers linked to Ethernet have an Ethernet controller (circuit board with specific Ethernet hardware). The Ethernet/GPIO board installed in the PS 300 logic cabinet provides the Ethernet controller for the PS 300.

Each computer in the network is physically tapped to the coaxial cable with a transceiver and transceiver cable (see Figure 1-1). Up to 100 computers can be linked to the same Ethernet, providing the distance between transceivers is at least 2.5 meters and the coaxial cable (or cable segments) is no longer than 500 meters (see Figure 1-2). Repeaters (shown in Figure 1-2 but not discussed in detail in this manual) are hardware devices used to amplify signals on the coaxial cable for larger Ethernet networks.

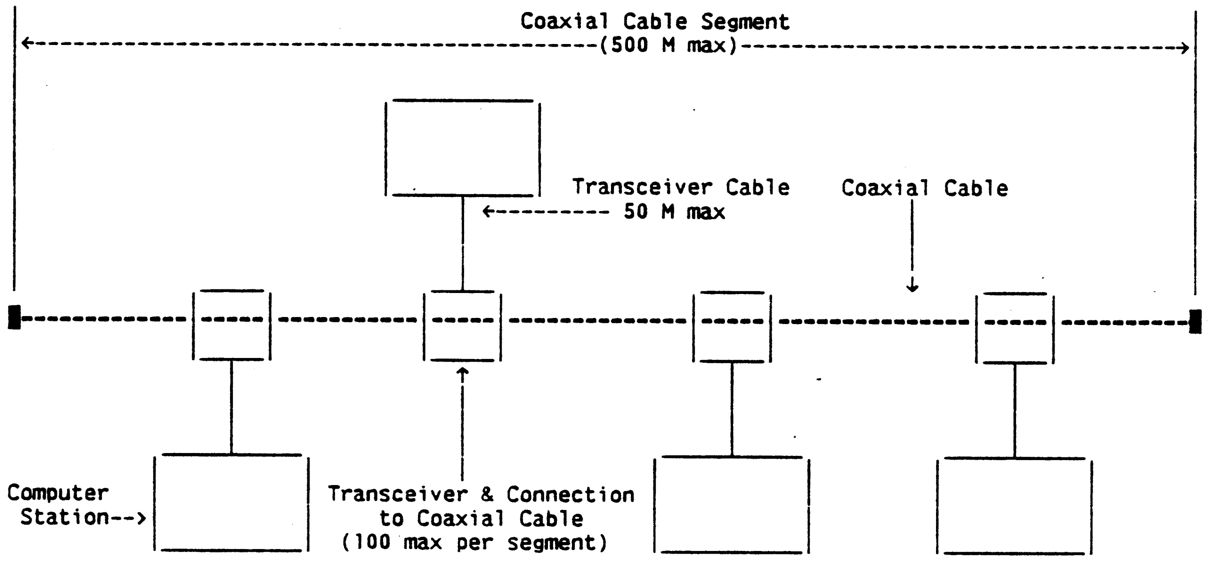


Figure 1-1. Minimal Configuration

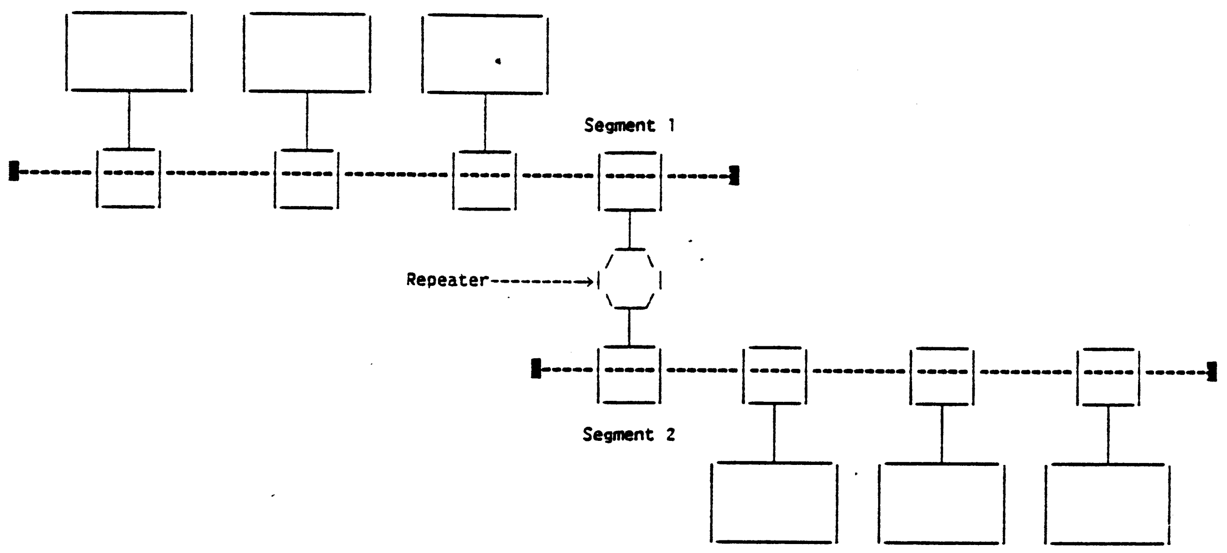


Figure 1-2. Medium Configuration

CUSTOMER-SUPPLIED SITE HARDWARE REQUIREMENTS

The customer must supply and install the following Ethernet hardware:

- Ethernet Host Controller Circuit Board
- Coaxial Cable
- Ethernet Transceiver
- Ethernet Transceiver Cable
- Ethernet Repeaters (if necessary)

As the customer, you must meet the following requirements before the hardware installation of the interface can take place.

- You should have a local area Ethernet network installed and running which meets the Ethernet specifications.

- Physical layer specifications of the Ethernet network include the installation of a coaxial cable connected to the host and transceivers (plus repeaters, if necessary) and transceiver cables for each computer station linked to the Ethernet coaxial cable.

- Coaxial cable and transceiver cable length must not exceed the maximum lengths set forth in the Ethernet specifications.
- You should install the transceiver and the transceiver cable but should not connect them to the PS 300.

  Ethernet specifications are documented in **The Ethernet: A Local Area Network; Data Link Layer and Physical Layer Specifications**. Version 2.0, November 1982. (Questions about Ethernet specifications or E&S-qualified vendors supplying Ethernet-compatible equipment should be directed to the area E&S account executive.)

For your reference, the hardware supplied by E&S for the PS 300 Ethernet interface consists of:

- Card Assembly, General Purpose Interface Option (GPIO) (E&S #204179-100)
- Cable Assembly, PS 300 Ribbon (E&S #204345-005)
- Screw (E&S #802004-103)
- Lock Washer (E&S #802300-008)

Retrofit Requirements

The PS 300 Ethernet interface requires a PS 300 system with a 2K Arithmetic Control Processor (ACP) card E&S #204130-101 (wire-wrap) A4 or above; or 204130-100 (PC) A3 or above.

The PS 300 PC Graphics Control Processor (GCP) card (E&S #204111-100, PC) must be at ECO revision level A3 or higher. The PS 300 wire-wrap GCP card (E&S #204111-101) must be at ECO revision level B0 or higher to support the PS 300 Ethernet interface. If ECOs are required to the GCP card, they must be completed prior to the installation of the option.

If the Ethernet interface to the PS 300 is to be installed in an existing system, make sure your cards are at the appropriate levels. If you do not know the level of your cards, contact the E&S Customer Service Center at 1-800-582-4375.

SITE SOFTWARE REQUIREMENTS

The PS 300 Ethernet interface currently runs only under VMS DECnet and UNIX operating systems on a DEC VAX. Separate firmware is required for each network protocol. Any implementation of the PS 300 Ethernet interface with any other operating system, network protocol, or host computer is not supported by Evans & Sutherland.

DECnet network software must be installed and running on the VAX before the E&S customer engineer arrives at the site to install the PS 300 Ethernet interface.

PS 300 Firmware/Software

As the customer, you are responsible for installing the E&S-supplied software on your host system. The standard firmware and software are included in the purchase price of the interface and are shipped to the software shipping address provided on the PS 300 sales checklist.

The following firmware/software packages must be available at your site prior to requesting installation. A packing list shipped with the software provides a complete list of the contents.

For PS 330 systems: PS 330 DECnet Ethernet Interface Package
E&S #904050-018

For PS 340H systems: PS 340H DECnet Ethernet Interface Package
E&S #904050-028

For PS 340S systems: PS 340S DECnet Ethernet Interface Package
E&S #904050-038

For PS 350 systems: PS 350 DECnet Ethernet Interface Package
E&S #904050-049

These packages contain the system firmware, interface software, diagnostic diskettes, Performance Verification Test, and demonstration diskettes.

E&S INSTALLATION PROCEDURES

When you have completed all the customer installation requirements, as shown on the customer installation checklist, you should request the hardware installation of the interface by contacting the E&S Customer Service Center at 1-800-582-4375. Before arriving on site, the customer engineer will verify that the system is ready for installation by going over the customer installation checklist with you.

At the installation, the host system manager (or someone with system privileges) must be present to oversee PS 300 Ethernet interface card installation and create the SITE.DAT file that is necessary for host/PS 300 communications.

Upon completion of the installation, the engineer will run interface diagnostics. These diagnostics verify the functionality of the Ethernet interface card.

After the diagnostics have been successfully run, the engineer will run the Performance Verification Test for the PS 300 Ethernet Interface (IPVT). This test will verify the communication link between the PS 300 and the host system. The communication link is tested by:

1. Loading a PS 300 function network by writing it via the Ethernet interface. The network manipulates the binary vectors.

2. Sending a binary vector list to the PS 300 via the Ethernet interface. The vector list can be manipulated by the function network in Step 1.

3. Performing a data recirculation test by sending various sized buffers of ASCII text to the host and then routing it back to the PS 300. The text is compared after recirculation.

The IPVT has run successfully when the binary vector list can be manipulated by the function network that activates the control dials and there are no errors logged by the data recirculation test.

Upon successful completion of all tests, you, as the customer, will sign a Performance Verification Test acknowledgement that initiates the 60-day product warranty.

2. HOST SOFTWARE

You are responsible for installing host-resident software on the host system prior to requesting installation of the interface hardware.

DESCRIPTION OF THE SOFTWARE

The software that supports the Ethernet Interface is distributed on 1600-bpi magnetic tape. The software contains the PS 300 Graphics Support Routines (GSRs) that contain the code necessary to communicate with the DECnet Ethernet Interface. These files should be read from the tape, compiled, and inserted in a library on the host. User application programs are linked with the GSRs library. The PS 300 GSRs are described in volume 3B of the **PS 300 Document Set**.

To read the GSR files off of the accompanying tape, the following DCL command sequence should be used:

```
$ ALLOCATE MTnn:          (MTnn: refers to the physical device name of
$ MOUNT  MTnn: PSDIST     the appropriate tape drive unit)
$ COPY Mtnn:*.*;* *
$ DISMOUNT MTnn:
$ DEALLOCATE MTnn:
$
```

The magnetic tape contains many other files that are not related to the Graphics Support Routines. Refer to the file README.TAP for a description of the contents of these other files. READFOR.GSR contains a short description of each of the FORTRAN GSR files.

INSTALLING THE FORTRAN GSRs

This section contains brief installation instructions for the DEC/VAX FORTRAN-77 version of the PS 300 GSRs. The GSRs will compile only under a FORTRAN-77 compiler and are supported under VMS Version 4.2 and higher. PS 300 Graphics Firmware Version A2.V01 or higher is required to run the Graphics Support Routines.

Table 2-1 lists the source files for the DEC VAX/VMS FORTRAN-77 version of the Graphics Support Routines, along with a description of each file.

The object module library containing ALL of the DEC VAX/VMS FORTRAN-77 GSR subroutines is contained in the file:

```
GSRF.OLB
```

To link your program with the DEC VAX/VMS FORTRAN-77 GSRs, enter the following command:

```
$ LINK <pgm>,<...any additional user object modules...>,GSRF/LIB
```

Table 2-1. FORTRAN-77 Graphics Support Routines

File Name	Description
GSRF.FOR	Source file for the GSRs.
PROFORLIB.FOR	Source file for the intermediate code between GSRF.FOR and PROLIB.MAR.
PROIOLIB.MAR	Macro source file for the low-level I/O subroutines used by the GSRs to communicate with the PS 300.
PROCOMF.FOR	Contains the global definitions of the FORTRAN-77 VAX GSR's and is INCLUDEd by GSRF.FOR.
PROFORCOM.FOR	Contains the global definitions for the PROFORLIB.FOR.
PROCONST.FOR	Contains file that may be INCLUDEd by the user in an application program. Contains the constant definitions.

The files CIRCLEF.FOR and BLKLEVF.FOR contain the source code of two VAX FORTRAN-77 GSR programs which demonstrate some of the subroutine calls. These files must be compiled and linked with the GSR library.

To recreate the DEC VAX/VMS FORTRAN-77 GSRs from the original source files, the following DCL command sequence should be used:

```
$ FORTRAN GSRF
$ FORTRAN PROFORLIB
$ MACRO   PROIOLIB
$ LIBRARY/CREATE GSRF GSRF,PROFORLIB,PROIOLIB
$
```

INSTALLING THE PASCAL GSRs

This section contains brief installation instructions for the DEC VAX PASCAL V2 version of the Graphics Support Routines (GSRs). The GSRs will compile only under a VAX PASCAL V2 compiler and are supported under VMS Version 4.2 and higher. PS 300 Graphics Firmware Version A2.V01 or higher is required to run the Graphics Support Routines.

Table 2-2 lists the source files for the DEC VAX/VMS PASCAL V2 version of the Graphics Support Routines, along with a description of each file.

Table 2-2. Pascal V2 Graphics Support Routines

File Name	Description
GSRP.PAS	Source file for the GSRs
PROPASLIB.PAS	Source file for the intermediate I/O procedures conceptually residing between GSRP.PAS AND PROIOLIB.MAR
PROIOLIB.MAR	Macro source file for the low-level I/O procedures used by the GSRs to communicate with the PS 300
PROCONST.PAS	File that should be INCLUDEd by the user in an application program. Contains CONSTant definitions.
PROTYPES.PAS	File that should be INCLUDEd by the user in an application program. Contains TYPE definitions.
PROEXTRN.PAS	File that should be INCLUDEd by the user in an application program. Contains EXTERNal definitions.

The object module library containing ALL of the DEC VAX/VMS PASCAL V2 GSR procedures is contained in the file:

```
GSRP.OLB
```

To link your program with the DEC VAX/VMS PASCAL V.2 GSRs, enter the following command:

```
$ LINK <pgm>,<...any additional user object modules...>,GSRP/LIB
```

The files CIRCLEP.PAS and BLKLEVP.PAS contain the source code of two VAX PASCAL V2 GSR programs which demonstrate some of the procedures. These files must be compiled and linked with the GSR library.

To recreate the DEC VAX/VMS PASCAL V2 GSRs from the original source files, the following DCL command sequence should be used:

```
$ PAS GSRP
$ PAS PROPASLIB
$ MACRO  PROIOLIB
$ LIBRARY/CREATE GSRP GSRP,PROPASLIB,PROIOLIB
$
```

ESTABLISHING HOST COMMUNICATION USING THE GSRs

The GRSs read a file called "PSDEVICE.DAT" to determine which device to access. Before a user program can run, this file must be generated with the following line:

```
LOGDEVNAM=nodename/PHYDEVTYP=ETHERNET
```

where nodename is the DECnet Node Name for the PS 300.

for example:

```
LOGDEVNAM=PS300A/PHYDEVTYP=ETHERNET
```

3. DECNET NODE ADDRESS--THE SITE.DAT FILE

As the customer, you are responsible for assigning and installing the PS 300 DECnet node address on the host computer and for installing this same address in the SITE.DAT file on the PS 300 firmware. These are both simple procedures.

The DECnet node address should be installed on the host at the same time the E&S-supplied software is installed, and prior to hardware installation. If this is a new system installation, the SITE.DAT file must be created after the hardware installation is complete, as the process requires access to the PS 300 floppy diskette drive.

NOTE

The GPIO/Ethernet card will not start processing network traffic until it receives a DECnet node address from the SITE.DAT file.

This chapter provides instructions for both procedures.

ASSIGNING THE PS 300 NODE ADDRESS ON THE HOST

The DECnet node address of the PS 300 must be known to the host before the PS 300 can be recognized. The DECnet node address is generally selected by the network manager or system manager for the host computer. The DECnet node address and name is then installed in the DECnet volatile and permanent data bases by the system manager. The following Network Control Program commands show how to install the node address of 1.46 (area = 1, node = 46) with the name of "PS300":

```
                1.2
NCP> set node 1.46 name PS300

NCP> define node 1.46 name PS300
                   1.2
```

See the **VAX/VMS Network Control Program Referance Manual** and the **Guide to Networking on VAX/VMS** for more details.

INSTALLING THE SITE.DAT

The SITE.DAT file is the final file on the PS 300 graphics firmware diskette. This file enables users to configure features for the PS 300 system in a bootable file. The file is assumed to contain a string of ASCII commands.

If this is a new system, the PS 300 control unit, display, and keyboard must be installed before you can create and download the SITE.DAT.

The information needed in the SITE.DAT file may be installed in one of two ways:

1. By downloading a host-resident file over an asynchronous line. This is the preferred method.

2. By accessing the command mode on the PS 300 keyboard and entering the information locally.

The DECnet node address for the PS 300 must be put in the SITE.DAT file of the PS 300. The command to put in the SITE.DAT file is

SEND 'xxxx' to <1>pi_o1$;
(Note that JCP A2.V02 users must send address to <1>**ei_o1$;**)

where xxxx is the DECnet node address in 4 hexidecimal digits. In the 16-bit DECnet node address, the most significant 6 bits are the area number and the least significant 10 bits are the node address in the above area. Note: If area numbers are not used on your network, the area number defaults to 1.

The algorithm to get the DECnet node address is therefore:

(area-number * 1024) + node-number

For example the DECnet node address for node 1.46 is

(1 * 1024) + 46 = 1070 decimal = 042E hexidecimal

Thus the SITE.DAT command would be

ei
SEND '042E' to <1>pi_o1$;

The program PSNODE described in appendix A can be used to convert a DECnet node address into the form for the SITE.DAT file.

Installing the SITE.DAT Using an Asynchronous Line

If you can communicate with the PS 300 over an asynchronous line, you must perform the following steps to install information in a SITE.DAT file that provides the PS 300 DECnet node address.

NOTE

The characters "↑\" represent the CTRL and backslash keys pressed simultaneously. This control character has an ASCII value of decimal 28.

1. Create a file on your host that contains the following:

```
↑\:     { control sequence that provides the routing byte  }
        { to write ASCII data to the firmware diskette      }

SEND 'xxxx' to <1>pi_o1$;        { xxxx is the 16-bit DECnet          }
                                 { address, in hexadecimal, assigned }
                                 { to the PS 300                      }

↑\;     { control sequence that provides the routing byte  }
        { to close the file on the firmware diskette        }

CLOSE SITE;    { the PS 300 command that closes the file }

↑\>     { control sequence that provides the routing byte            }
        { to route following characters to the terminal emulator  }
```

For example, the DECnet address of the node represented by 1.46 described earlier would be installed in the SITE.DAT as:

```
↑\:
SEND '024E' to <1>pi_o1$;
↑\;
CLOSE SITE;
↑\>
```

2. Make a backup copy of the PS 300 graphics firmware that supports your system. Instructions for copying the firmware are in Volume 5 of the **PS 300 Document Set**. For new systems, the copy can only be made after the successful installation of the E&S system hardware and the completion of the PS 300 Performance Verification Test.

3. Mount the backup copy of the firmware diskette and boot the system. Booting instructions are provided in Volume 1 of the **PS 300 Document Set.**

4. Copy the host-resident SITE.DAT file from the host to the PS 300 diskette using the asynchronous line and standard host-system utilities.

5. Reboot the system using the diskette that contains the newly created SITE.DAT.

Installing the SITE.DAT Locally Using the PS 300 Keyboard

If you cannot communicate with the PS 300 over an asynchronous line, the SITE.DAT file information must be entered directly from the PS 300 keyboard. To install information in a SITE.DAT file that provides the PS 300 node address, do the following:

1. Make a backup copy of the PS 300 graphics firmware that supports your system. Instructions for copying the firmware are in Volume 5 of the **PS 300 Document Set**. For new systems, the copy can only be made after the successful installation of the E&S system hardware and the completion of the PS 300 Performance Verification Test.

2. Mount the backup copy of the firmware diskette and boot the system. Booting instructions are provided in Volume 1 of the **PS 300 Document Set.**

3. Access command mode on the PS 300 keyboard by pressing the CTRL/LINE LOCAL keys simultaneously. The command prompt "@@" should appear.

4. Enter the following commands (each command line must be followed by a RETURN):

```
Configure A;
SEND 'Send ' 'xxxx' ' to <1>pi_o1$;' to <1>Wda0;
SEND 'Close Site;' to <1>Wdac0;
Finish Configuration;
```

Note the occurrence of two single quotes before and after xxxx.

In the above command sequence, xxxx is the 4-digit hexadecimal DECnet node address assigned to the PS 300 preceded and followed by pairs of single quotes. For example, the DECnet node address of node represented by 1.46 described earlier would be installed in the SITE.DAT as:

```
Configure A;
SEND 'Send ' '042E' ' to <1>pi_o1$;' to <1>Wda0;
SEND 'Close Site;' to <1>Wdac0;
Finish Configuration;
```

NOTE

If you enter a command incorrectly from the keyboard, and the command **'Close Site;'** has not been sent to <1>Wdac0, you can reboot the system and start over. However, if the **'Close Site;'** command has already been sent to Wdac0, you must delete the SITE.DAT file that now exists on the firmware diskette before you reboot and begin again. Instructions for deleting files on the firmware diskette are provided in Volume 5 of the **PS 300 Document Set.**

5. Reboot the system using the diskette that contains the newly created SITE.DAT file.

4. PS 300 ETHERNET COMMUNICATION PROTOCOL

The PS 300 DECnet Ethernet GPIO Interface was designed so that any computer host that supports DECnet, VAX Version 4.0, can establish one high speed link to a PS 300 graphics terminal.

ETHERNET CONFIGURATION PROTOCOL

The GPIO supports the Ethernet Version 2.0 configuration protocol to enable testing of the network. Refer to the **Guide to Networking on VAX/VMS.**

DECnet Routing Layer

The GPIO operates as a DECnet Ethernet End Node. Therefore, at least one other node on the Ethernet must be a Routing Node.

DECnet NSP Layer

The GPIO supports the full implementation of the NSP Protocol, Version 4.0.0, with the following exceptions:

1. Only one logical connection from a host to the PS300 is supported at any one time.
2. Since the PS 300 is considered a slave device to the host, the PS 300 cannot request the GPIO to establish a connection link to a host.
3. Since Session Control Message Flow Control has been made obsolete by DEC, it is not implemented in the GPIO.

Network Management Layer

The PS 300 does not provide support for any Network Management or Maintenance functions except for the Ethernet Configuration Protocol.

PART II

USER INFORMATION

5. PS 300 DISPLAY STRUCTURES

This chapter describes the PS 300 display structures. Information in this chapter supports the physical I/O capabilities of the interface. This information is provided for advanced PS 300 programmers.

OVERVIEW

Display structures in the PS 300 represent the operations and data that form the two- and three-dimensional objects constructed by user application programs. The display structures are traversed each refresh cycle by the display processor. These structures are contained in a structured display file which is created and modified under control of the Graphics Control Processor (GCP), the 68000 processor in the system.

NODE DESCRIPTION

Display structures are organized as an acyclic hierarchy of nodes that are either:

- Operation nodes that change the "state of the machine" for descendent data nodes.
- Data nodes (dots, lines, polygons, or characters).
- Instance nodes, sometimes known as set nodes, that group lists of branches of the acyclic hierarchy that are to be traversed.

Every type of node may be named. Naming a node causes the name to be entered into a dictionary table (hash table), along with a pointer indicating where the node associated with that name resides in PS 300 mass memory. The name in the hash table is located in what is called an alpha block. The address of the alpha block remains constant as long as the named node is not deleted or referenced by another node (or function) in the system.

When a node in the system references another node, an alpha pointer is placed to the alpha block to link the current node with that name. This means that a level of indirection is introduced for every reference to an entity in the system. The indirection is a small penalty for the flexibility this procedure provides. It also implies that there is an alpha block for every node, regardless of whether the user has chosen to associate a name with the node.

The display structures are constantly being traversed for display by the display processor. When a node is changed, the GCP makes a copy of the changed node, changes the elements, and then changes the pointer to the node in the alpha block. This occurs whenever a matrix is changed, a new display is enabled, etc. In essence, then, every change to the display structures causes the change to be "double buffered" until the display processor makes the change at the end of each refresh cycle. If the GCP changes that data structure, a named entity is never in the same location in mass memory. Conversely, the PS 300 never performs "garbage collection" on existing named entities.

If the node is never referenced by a function network (or externally from the host) a named entity is always in the same location in mass memory. This permits the physical I/O capabilities to provide extremely close coupling of the host and PS 300 display structures (not the PS 300 system).

The Lookup Named Entity I/O function implemented by the PS 300 device driver returns the actual mass memory address of the node rather than the pointer to the alpha block. This means that values in the node can be changed directly. However, if the node is being displayed, the ACP may traverse the node during the brief but finite time frame when are being changed. This may result in an improper picture being displayed (new x value with old y value; or a matrix with mixed values -- part new, part old), but should never cause the display processor to traverse the display structures improperly as long as no pointers are changed. (Pointers should never be changed with the physical I/O facilities; that is best left to the GCP).

Thus, given the exact formats of the display structures and the Read Physical and Write Physical I/O requests, display structures can be written directly under host control (without any interference by the GCP) as long as the named entities to be updated are:

1. Created initially by the GCP using standard PS 300 commands.

2. Not involved in any way with local operations (function networks). However, see NOTE below.

3. Not changed by the host program in any way except by using the physical I/O facilities.

NOTE

Nodes that precede nodes to be updated using the physical I/O capability can be involved with local operations. For example, viewing operations can be performed locally while updating modeling transformations using the physical I/O facility.

Most often only matrices and/or data nodes are modified using the physical I/O requests. However, all named entities can be modified in the same manner given the same assumptions. The exact data formats for some operation nodes (matrices) and data nodes follow.

Operation Nodes

An operation node is a data structure element that modifies the state of the display processor. An operation node consists of:

- an integer that indicates the display structure is an operation node (=1).
- an integer that specifies the particular type of operation node, the descendant alpha, and a variable number of fields required by the particular type of operation node.

Because an operation node modifies the state of the Arithmetic Control Processor (ACP), the ACP state must be saved before traversing a hierarchical branch which includes an operation node.

The state is then restored before traversing the next hierarchical branch. For any operation node, bit 15 of the operation type is a conditional bit. If this bit is set (and if bit 15 [the blink bit] in the Condition Mask of the ACP State is zero), the associated operation node is not executed. In all other cases, the operation node is executed. In all cases, descendent operation nodes are traversed.

NOTES

1. Each element of the following blocks of data is a 16-bit element. Double elements, shown as "|--   --|", are 32-bit elements.

NOTES (continued)

2. In the 32-bit mantissas (fractions) used for translates that are shown as:

```
|-------------------------------------|
|--            Tx(H)                --|
|              Tx(L)                  |
|-------------------------------------|
|--            Ty(H)                --|
|              Ty(L)                  |
|-------------------------------------|
|--            Tz(H)                --|
|              Tz(L)                  |
|-------------------------------------|
```

   the most significant bit of the second word should always be 0 (zero), as shown in the figure below. This is also true for the PS 350 32-bit mantissas.

```
|-------------------------------------|
|--            Tz(H)                --|
|0             Tz(L)                  |
|-------------------------------------|
```

   This also applies to the matrix mantissas show as:

```
|-------------------------------------|
|           M [1,1(H)]                |
|--                                 --|
|           M [1,1(L)]                |
|-------------------------------------|
```

   where the most significant bit of the second word should always be 0 (zero), as shown in the figure below.

```
|-------------------------------------|
|           M [1,1(H)]                |
|--                                 --|
|0          M [1,1(L)]                |
|-------------------------------------|
```

3. A "C" in the left corner of the Operation Type field indicates the Conditional Bit (bit 15).

The following figures provide the formats for the operation nodes.

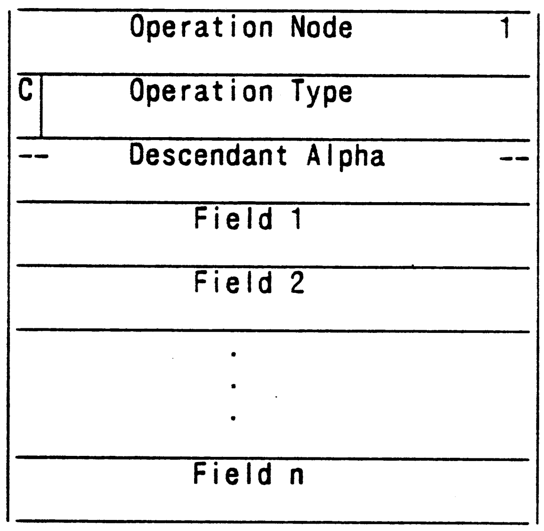


Figure 5-1. General Operation Node Format

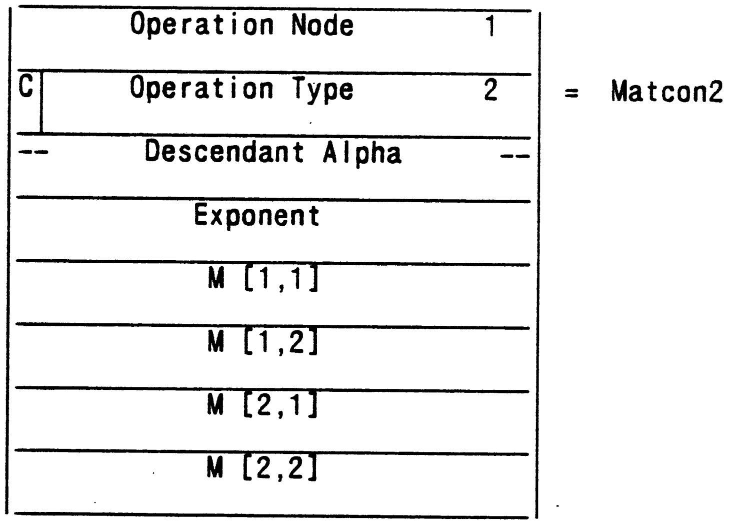


Figure 5-2. Operation Node - Matrix Concatenation 2x2 (Matcon2)

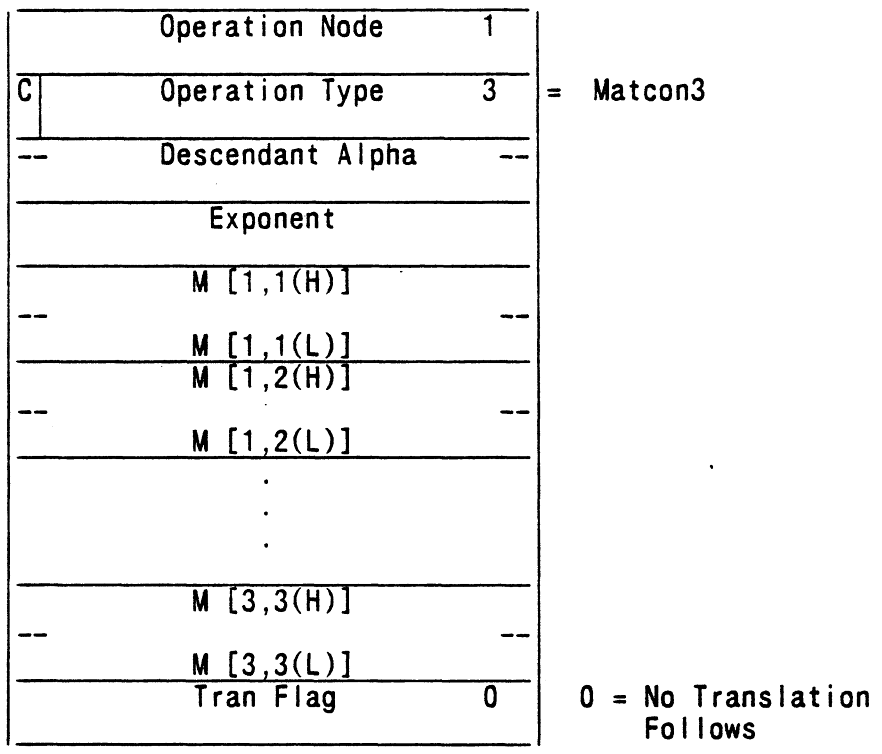


Figure 5-3. Operation Node - Matrix Concatenation 3x3 (Matcon3)

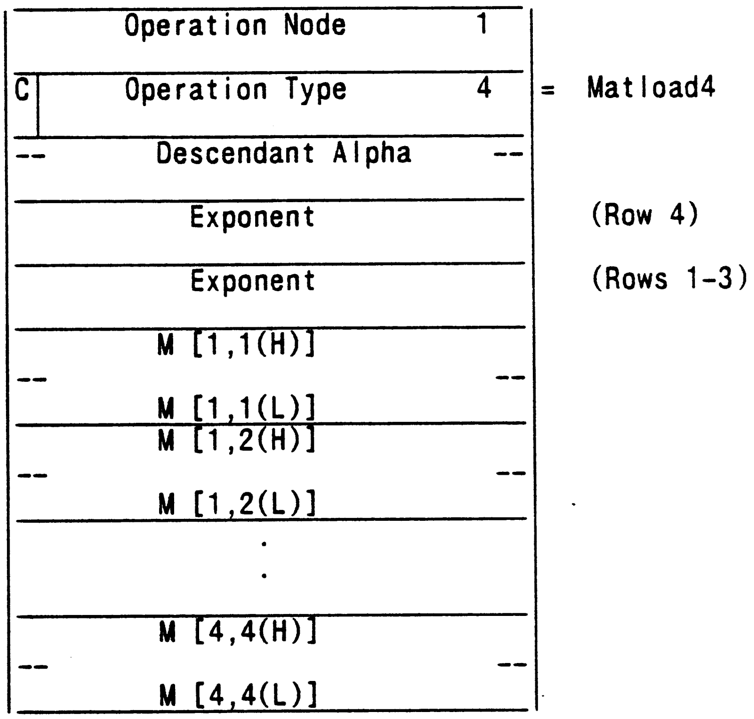


Figure 5-4. Operation Node - Matrix Load 4x4 (Matload4)

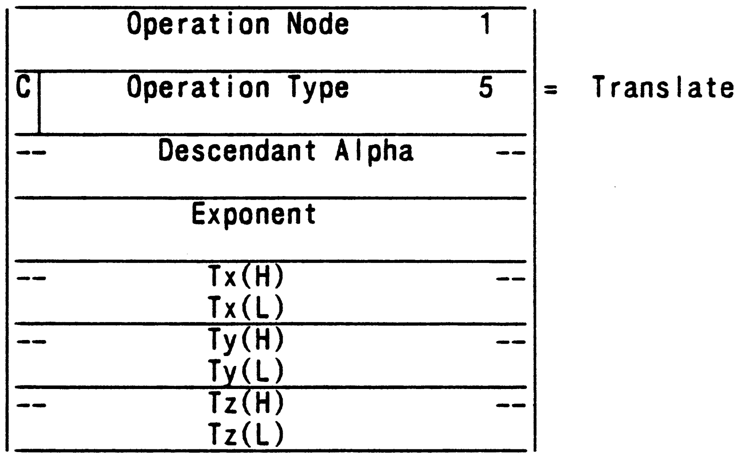


Figure 5-5. Operation Node - Translate

	Operation Node	1	
C	Operation Type	10	= Mat3 Trans
--	Descendant Alpha	--	
	Exponent		
--	M [1,1(H)] M [1,1(L)]	--	
--	M [1,2(H)] M [1,2(L)]	--	
	. .		
--	M [3,3(H)] M [3,3(L)]	--	
	Tran Flag	1	1 = Translation Follows
	Exponent		

--	Tx(H)
	Tx(L)
		--	

--	Ty(H)
	Ty(L)
		--	

--	Tz(H)
	Tz(L)
		--	

Figure 5-6. Operation Node Matrix Concatenation 3x3 and Translate Concatenation 1x3

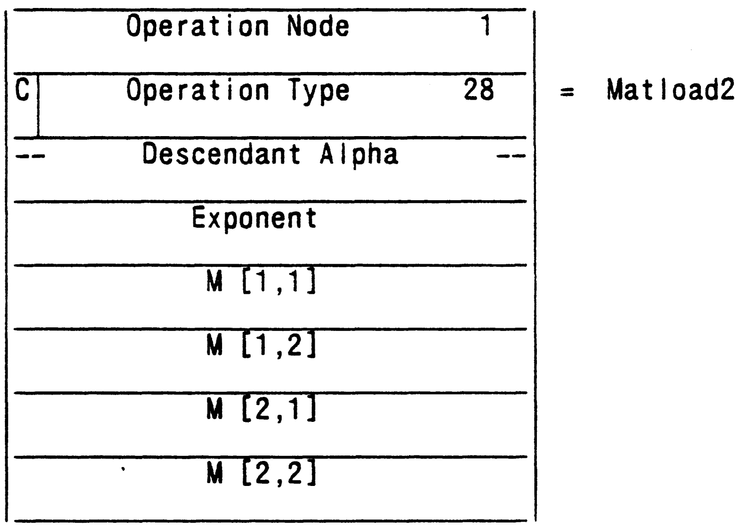


Figure 5-7. Operation Node - Matrix Load 2x2 (Matload2)

Data Nodes

A data node is the display structure primitive that causes the display processor to convert data into a picture. A data node consists of:

- An integer that indicates this display structure is a data node (=2).
- An 8-bit field that specifies the mode of vectors in the data node.
- An 8-bit integer that specifies the particular type of data node.
- A 32-bit integer which points to the next data node of identical data type.
- An integer (n) that specifies the number of vectors, polygons or characters in the data node.
- A 16-bit integer that specifies the pick index.
- Either vector data (including polygons) or character data.

Vector data consist of the two- or three-dimensional vectors (preceded by polygon attribute information if the data are polygons).

Character data consist of an initial translation, spacing information, and the character string.

Figure 5-8 shows the general data node format.

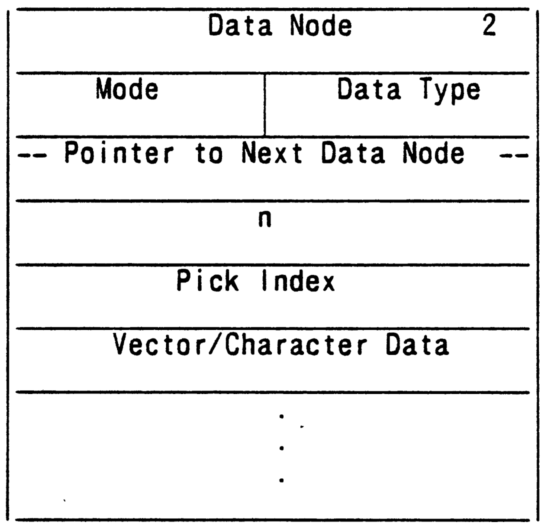
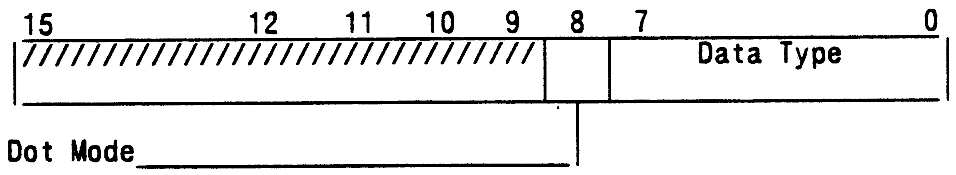


Figure 5-8. General Data Node Format

Mode, data type, pointer to next data node, pick index, and vector/character data are detailed further below.

Mode Field

The mode field of a data node consists of:



Dot Mode = 0 for no endpoint intensification
Dot Mode = 1 for endpoint intensification

The dot mode field of a data node is a single bit that specifies how the vectors are to be drawn. When dot mode = 0, vectors are drawn normally. When dot mode = 1, each endpoint of the vector list is drawn as an intensified dot.

Data Type

The data type field specifies the particular format of the data node. The display processor accepts vectors of two formats:

1. Vector-normalized data (full vectors).

   Vector-normalized data consist of 16-bit, signed binary fractions that share a common 8-bit, signed integer exponent and an explicit 7-bit, intrinsic intensity for each vector. For the vector:

   (x,y,z,i): x=2e*fx, y=2e*fy, z=2e*fz, i=i

   where e is the signed 8-bit integer exponent; the 16-bit significant digit fields fx, fy, and fz satisfy $-1<f<1$; and the 7-bit intrinsic intensity field i satisfies $0<i<1$.

2. Block-normalized data

   Block-normalized data consist of 16-bit signed binary fractions that share a common 7-bit signed integer exponent and an explicit 8-bit intrinsic intensity for each block of vectors. For the vectors:

   x1 = 2e*fx1, y1 = 2e*fy1, z1 = 2e*fz1, i=i
   x2 = 2e*fx2, y2 = 2e*fy2, z2 = 2e*fz2, i=i
   .
   .
   .
   xn = 2e*fxn, yn = 2e*fyn, zn = 2e*fzn, i=i

   where e is the signed 8-bit integer exponent; the 16-bit significant digit fields fx, fy, and fz satisfy $-1<f<1$; the 7-bit intrinsic intensity field i satisfies $0<i<1$.

   Block-normalized data may be treated as 16-bit fixed point data by applications that (1) only require 16 bits of precision and (2) wish to avoid converting integers to PS 300 data formats.

3. Polygon Data

   Vectors contained within polygon nodes consist of 3D vector-normalized data as described above.

4. Character Data

   Character data consist of a character string, an initial translation that positions the character string, and information that controls the spacing between characters.

Next Data Node Field

The next data node field contains a 32-bit pointer to the next data node of identical type (0 = nil pointer). This pointer allows a set of character strings to be grouped together (Label Block). It also replaces the need to have a VecSet node group a set of vector lists together.

Pick Index Field

The pick index field of a data node is reported with the vector count when a pick occurs, identifying the vector list in which the pick occurred.

Although the number of vectors that may be contained in a data node is 65,535 (if n is treated as a 16-bit unsigned number), by convention the maximum number of vectors that are specified in a given data node block is 2,048. (The actual number of vectors is usually much smaller to avoid memory fragmentation.) This is less than the maximum number of vectors that may be counted during pick processing.

The software that creates data nodes ensures that the index is correct for a given data node and that the reported index (together with the vector count) allows the vector that was picked to be correctly identified.

Vector Data

All vector data processed by the display processor are numbers of normalized, floating-point form, such as 2e*f, where e is a signed-integer exponent and the significant digit field, f, satisfies $-1<f<1$. Rather than provide an exponent for each coordinate of a vector, the display processor associates a single exponent with each vector or block of vectors.

All vector data are two- or three-dimensional (i.e., x,y or x,y,z), with an implicit, homogenous coordinate equal to 1 (i.e., x,y,z,1). The dynamic range gained by explicit use of the homogenous coordinate is provided by representing vector data in the normalized, floating-point form.

Polygon Data

Polygon nodes contain vectors that define the polygon as well as a pointer (ptr) to another node describing the attributes of the polygon.

In addition, polygon vectors have implicit closure; that is, there is an implied vector from the last point of the polygon to the first point. The ACP automatically displays this implied vector.

Character Data

Character data consist of a character string, an initial translation that positions the character string, and information that controls the spacing between characters.

The initial translation consists of 16-bit, signed binary fractions for x, y, and z, with an implicit, homogeneous coordinate equal to 1 (i.e., x,y,z,1), and a shared 8-bit, signed integer exponent. Thus, the translation:

x,y,z,1   x=2e*fx, y=2e*fy, and z=2e*fz,

where e is the signed, 8-bit integer exponent, and where the 16-bit significant digit fields fx, fy, and fz satisfy $-1 <= f < 1$.

The spacing information consists of a delta x and a delta y, each a 16-bit, signed binary fraction, sharing an implied exponent equal to zero. The delta x and delta y values determine the separation between characters in the x and y directions. They are given in the coordinate space of the characters themselves, satisfying the range: -1 <= delta x, delta y < 1.

For each character in a string of characters, the corresponding character stroke block is read from mass memory to provide the vectors which make up the individual character.

The following figures provide descriptions of the data nodes for:

Vector-normalized (Full Vector) 3D (Vec3f0) data node
Vector-normalized (Full Vector) 2D (Vec2f0) data node
Block-normalized 3D (Vec3b0) data node
Block-normalized 2D (Vec2b0) data node
Polygon (Vecpoly) data node
Character (Dchar) data node

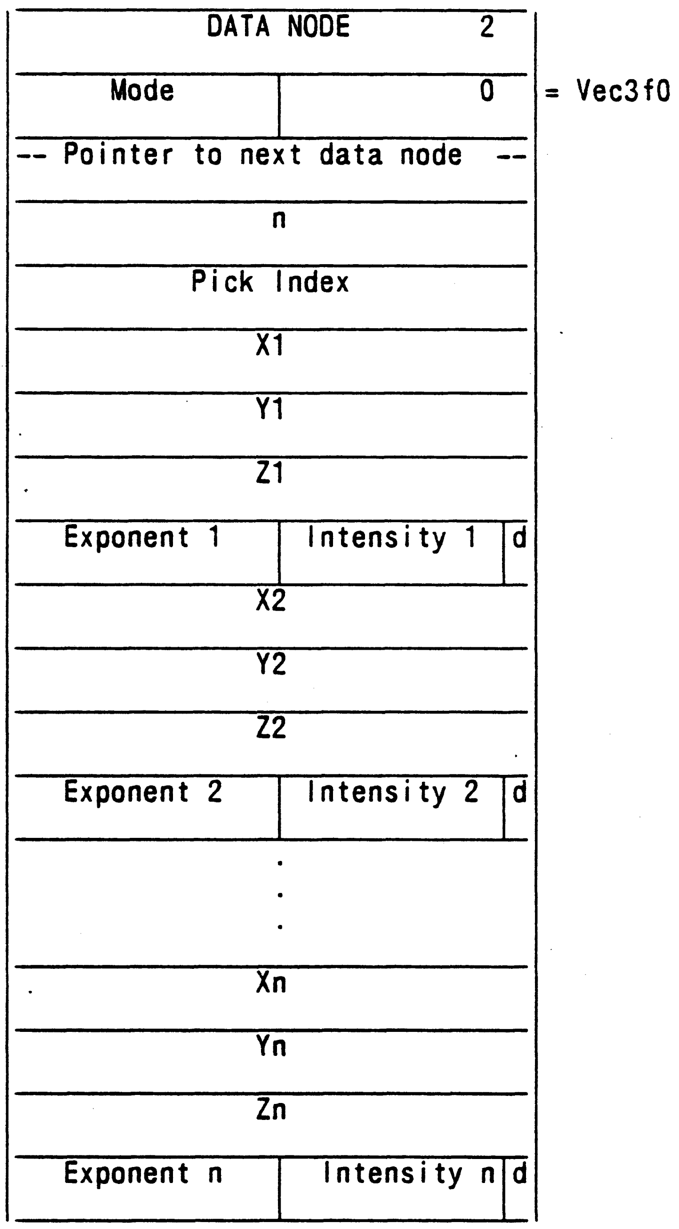


Figure 5-9. Vector-Normalized (Full Vector) Data Node - 3D (Vec3f0)

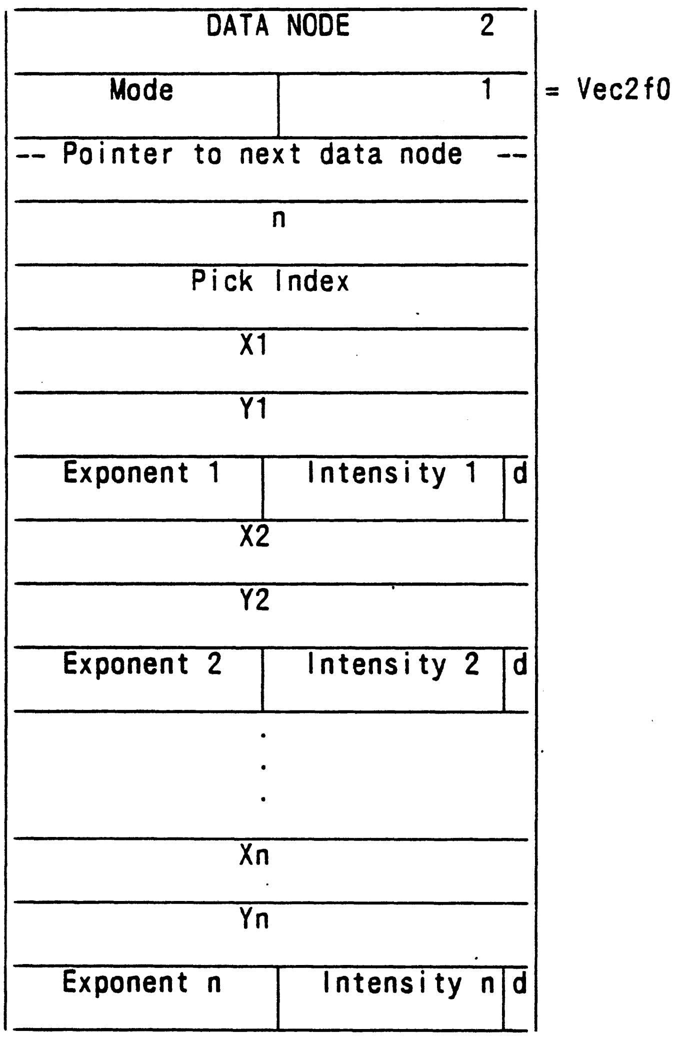


Figure 5-10. Vector-Normalized (Full Vector) Data Node - 2D (Vec2f0)

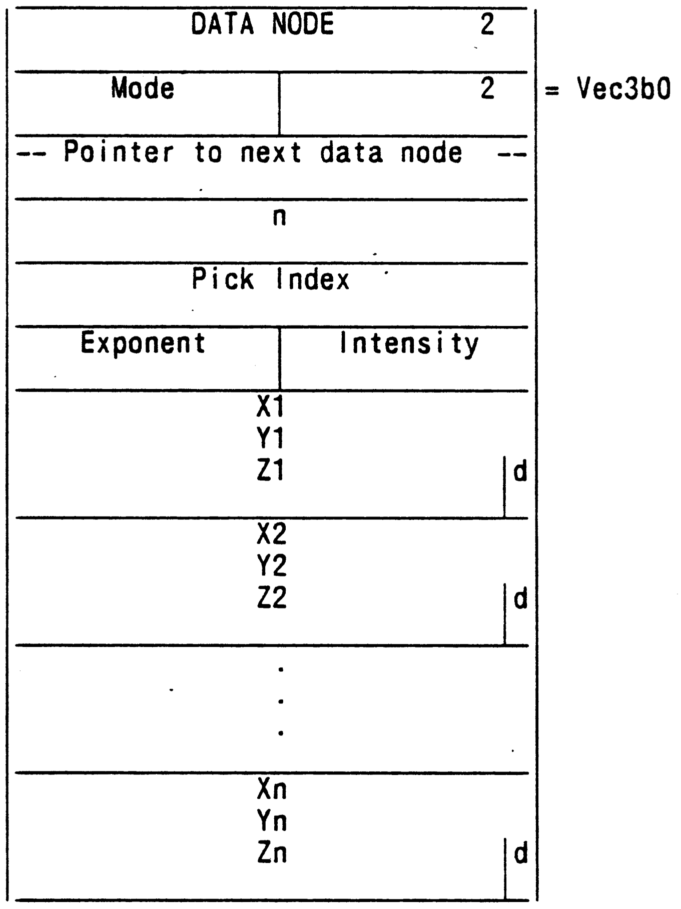


Figure 5-11. Block-Normalized Data Node - 3D (Vec3b0)

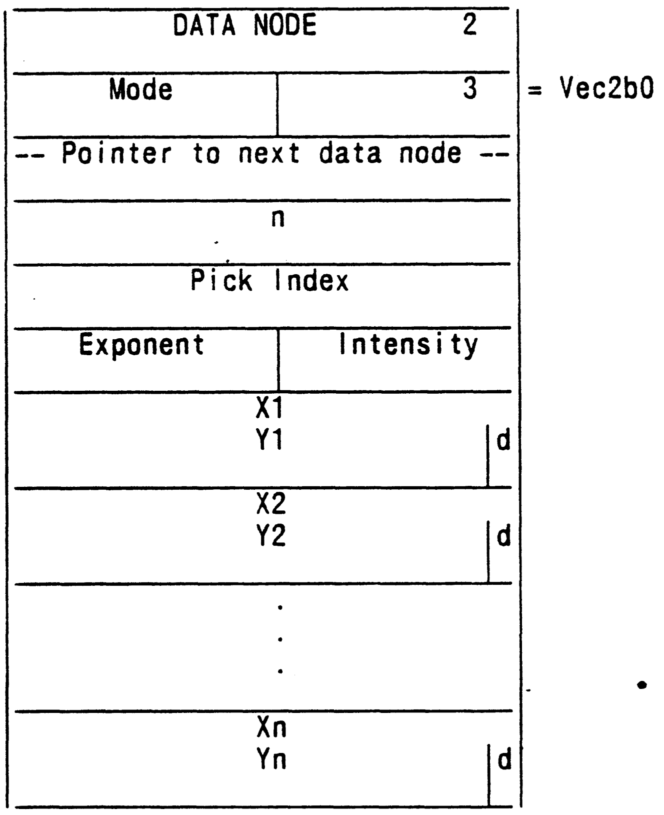


Figure 5-12. Block-Normalized Data Node - 2D (Vec2b0)

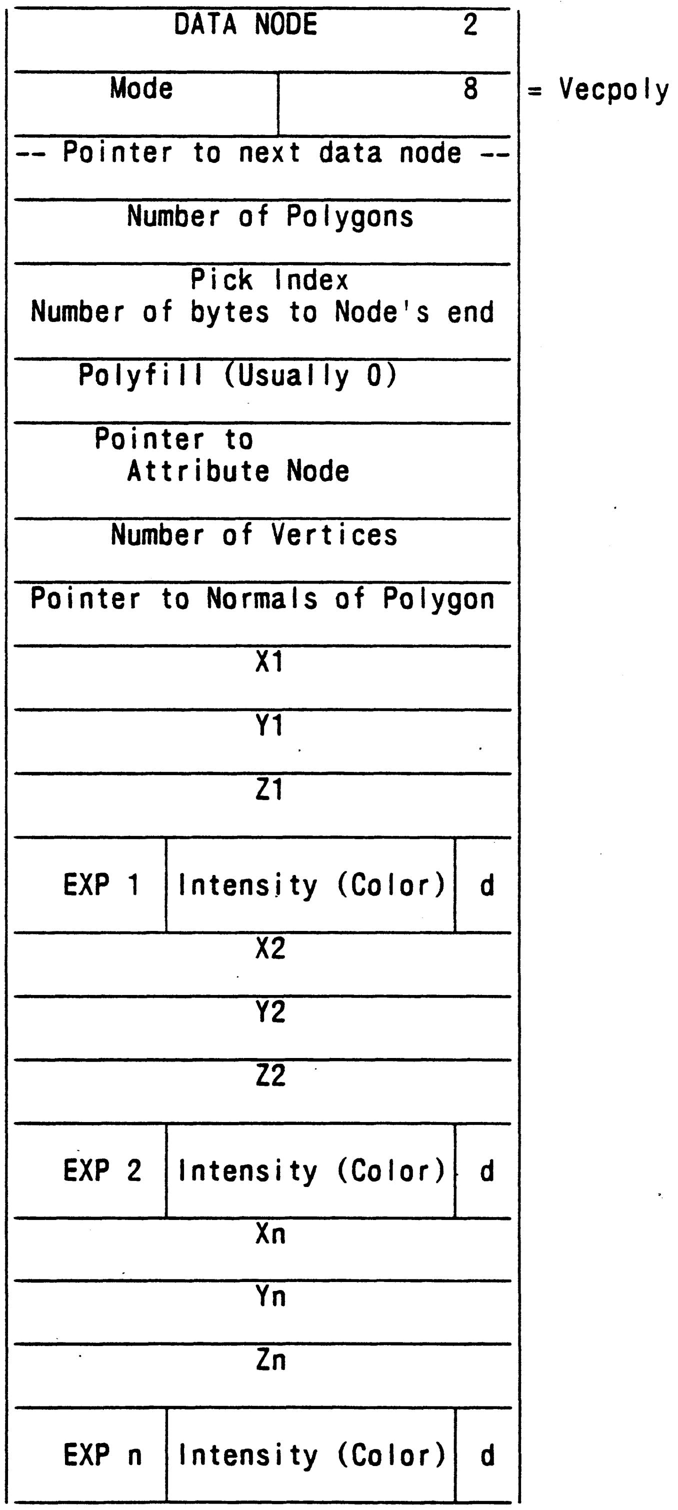


Figure 5-13. Polygon Data Node (Vecpoly)

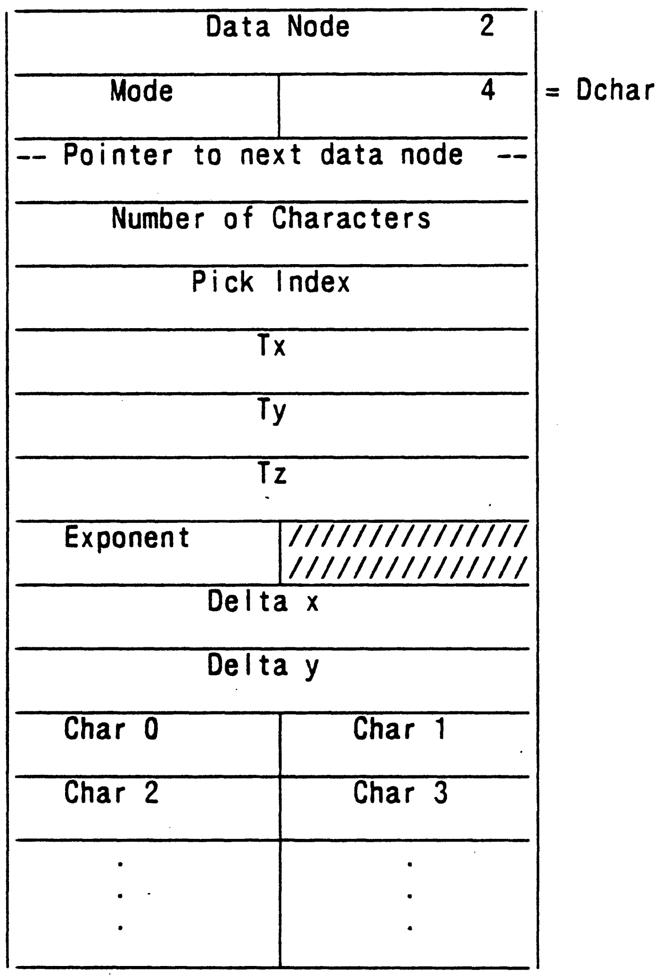


Figure 5-14. Character Data Node (Dchar)

ADVANCED PHYSICAL I/O PROGRAMMING

The physical I/O process can produce distorted pictures when it is updating display structures at the same time the display processor is traversing them.

To avoid this "single buffer" phenomenon, these display structures can be "double buffered." This is done by creating two copies of the named entities to be updated with different names (like Data1 and Data2). The data structures are then alternately updated and displayed using either the IF LEVEL_OF_DETAIL or IF CONDITIONAL_BIT commands such as:

```
IF LEVEL = 1 THEN Data1;
IF LEVEL = 2 THEN Data2;
```

or

```
IF BIT 0 ON THEN Data1;
IF BIT 0 OFF THEN Data2;
```

These commands are used in conjunction with a node higher in the structured display file that either sets the level of detail (SET LEVEL) or sets the conditional bit (SET BIT).

The node that performs the SET BIT and SET LEVEL operation is the change bits operation node and is shown in Figure 5-15. This operation node is also used to set displays, set character orientation, set contrast, set CSM, set depth clipping, set plotter, set rate external, set blinking (PS 350 only), and set line texture (PS 350 only).

```
Wordindex = 0 - LOD value
            1 - Conditional bits
```

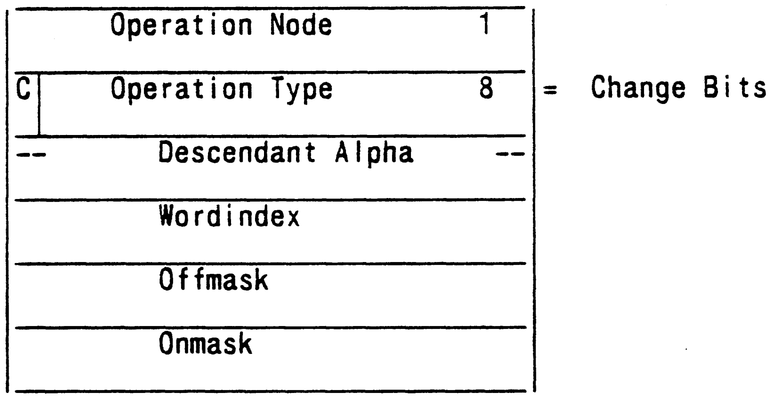


Figure 5-15. Change Bits Operation Node

This SET LEVEL or SET BIT node is updated using the physical I/O to "swap buffers." The physical I/O Write command updates multiple blocks of data in mass memory in one I/O operation (including the change to the SET LEVEL or SET BIT node as the last operation) and ensures that the buffers are swapped on the next refresh.

The physical I/O Write/Sync operation ensures that each buffer gets at least one refresh before allowing the next write operation.

The ACP turns off the bits specified in OFFMASK. The ACP then turns on the bits specified in ONMASK.

RAWBLOCK

The RAWBLOCK command is used to allocate memory that can be directly managed by a user-written function or by the physical I/O capabilities of the PS 300.

The command:

```
<name> := RAWBLOCK i;
```

carves a contiguous block of memory such that there are "i" bytes available for use. Since this has to be a display structure and one contiguous memory, it is structured so that it looks like Figure 5-16.

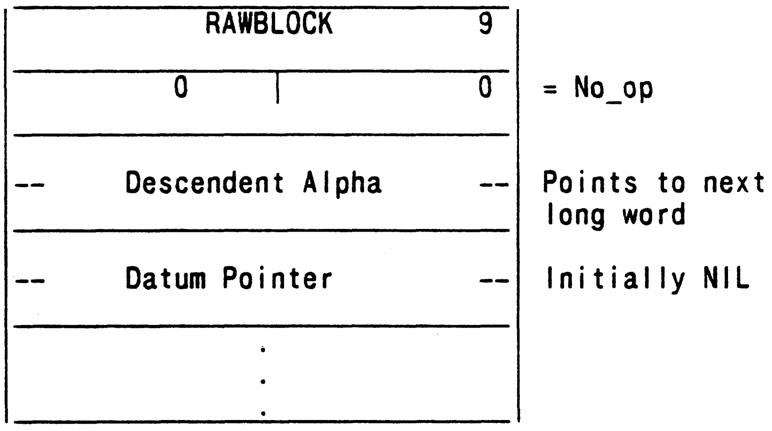


Figure 5-16. Rawblock Data Node

The block looks like an operation node to the ACP. The descendent alpha points to the next long word in the block. What the ACP expects in this word is the .datum pointer of the alpha block. (The .datum pointer points to the first structure to be traversed by the ACP. This is the address in memory where the data associated with a named entity is located.)

To use this block, the physical I/O operation (or a user-written function) fills in the appropriate structure following the .datum pointer. When this is complete, it changes the .datum pointer to the proper value and points to the beginning of the data. After the ACP examines this structure, it displays the newly-defined data. Use the ACPPROOF procedure to change the .datum pointer with a user-written function. For more information on user-written functions, refer to "User-Written Function Facility" in the **PS 300 Document Set**, Volume 4.

When changing the .datum pointer using physical I/O, write the first (high-order) word to 0, write the correct (low-order) word, and then write the correct first (high-order) word. This prevents the display processor from interpreting the .datum field as a wrong pointer. (The ACP microcode interprets a high-order word as 0 and a NIL pointer, regardless of the contents of the low order word.)

More than one data structure at a time can exist in a RAWBLOCK. It is up to the user to manage all data and pointers in a RAWBLOCK.

A RAWBLOCK may be displayed or deleted like any other named data structure in the PS 300 (e.g., DISPLAY "name"; or DELETE "name";). When a RAWBLOCK is returned to the free storage pool, the PS 300 firmware recognizes that it is a RAWBLOCK and does not delete any of the data structures linked to RAWBLOCK.

PS 350 DISPLAY STRUCTURES

The following section contains the new data formats for the PS 350 graphics system. These formats are provided for users with parallel interface capabilities and for those users who access internal data in the PS 350.

Asterisked fields are neither used nor accessed by normal ASCII and GSR commands. The top bit in the second word of each of these formats (labeled "A") is a flag which, if clear, tells the display structure walker to process these fields. This bit is set by default and there exists no command to clear it. However, functions and programs using physical read/write facilities may choose to make use of these fields.

Vec3bd0

Vec3bd0 are 3D vectors, block-normalized, with 32-bit precision mantissas for x, y, and z. They give greater precision when displayed than the standard PS 330 vectors because the concatenation with the transformation matrix is done in double precision and their mantissas are 32 bits.

NOTE

Mantissa values are given in a peculiar format: The low-order word of each 32-bit value is shifted down one bit, with a leading zero added in bit position 15.

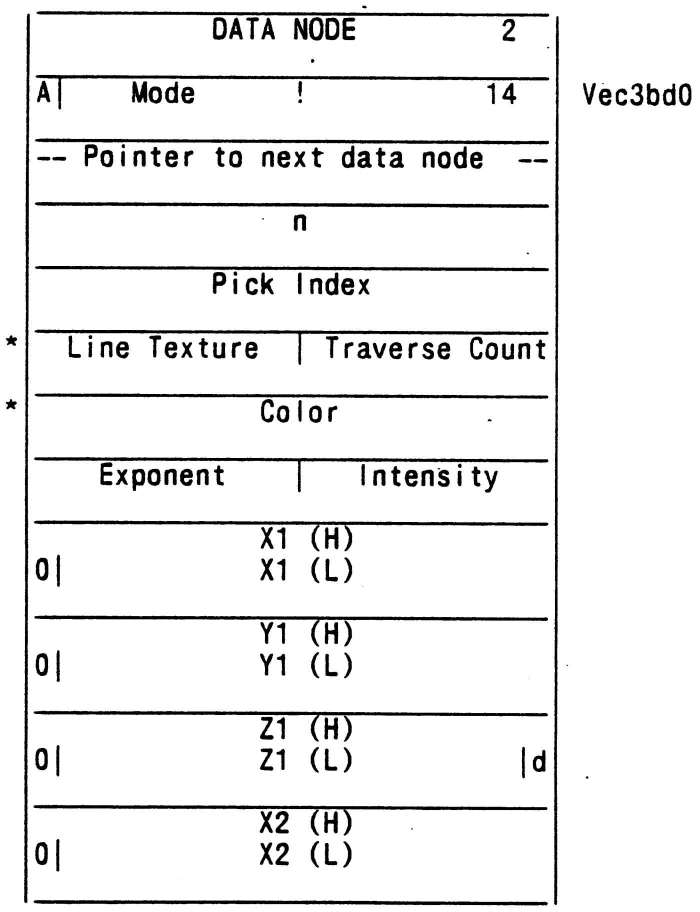


Figure 5-17. Block-Normalized Data Node - 3D (Vec3bd0)

(continued next page)

(continued from previous page)

```
|                                            |
|--------------------------------------------|
|                 Y2 (H)                     |
|0|               Y2 (L)                     |
|--------------------------------------------|
|                 Z2 (H)                     |
|0|               Z2 (L)                  |d |
|--------------------------------------------|
|                   .                        |
|                   .                        |
|                   .                        |
|--------------------------------------------|
|                 Xn (H)                     |
|0|               Xn (L)                     |
|--------------------------------------------|
|                 Yn (H)                     |
|0|               Yn (L)                     |
|--------------------------------------------|
|                 Zn (H)                     |
|0|               Zn (L)                  |d |
|--------------------------------------------|
```

Figure 5-17. Block-Normalized Data Node - 3D (Vec3bd0)

Double precision vectors, double precision matrix multiply.

Vec2bd0

Vec2bd0 are 2D vectors, block-normalized, with 32-bit precision mantissas for x, and y. They give greater precision when displayed than the standard PS 330 vectors because concatenation with the transformation matrix is done in double precision and mantissas are 32 bits long.

NOTE

Mantissa values are given in a peculiar format: The low-order word of each 32-bit value is shifted down one bit, with a leading zero added in bit position 15.

```
  ---------------------------------------
  |         DATA NODE               2   |
  |-------------------------------------|
  |A|     Mode      |              15   |   Vec2bd0
  |-------------------------------------|
  |-- Pointer to next data node      -- |
  |-------------------------------------|
  |                 n                   |
  |-------------------------------------|
  |             Pick Index              |
  |-------------------------------------|
* |  Line Texture   | Traverse Count    |
  |-------------------------------------|
* |               Color                 |
  |-------------------------------------|
  |    Exponent     |   Intensity       |
  |-------------------------------------|
  |                X1 (H)               |
  |0|              X1 (L)               |
  |-------------------------------------|
  |                Y1 (H)               |
  |0|              Y1 (L)            |d |
  |-------------------------------------|
  |                X2 (H)               |
  |0|              X2 (L)               |
  |-------------------------------------|
  |                Y2 (H)               |
  |0|              Y2 (L)            |d |
  ---------------------------------------
```

Figure 5-18. Block-Normalized Data Node - 2D (Vec2bd0)

(continued next page)

(continued from previous page)

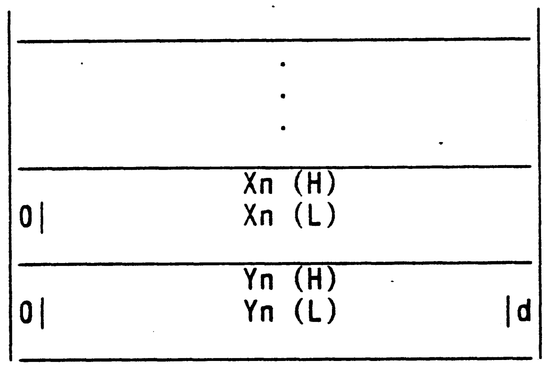


Figure 5-18. Block-Normalized Data Node - 2D (Vec2bd0)

Double precision vectors, double precision matrix multiply.

Vec3bs2

Vec3bs2 are 3D vectors, block-normalized, with 16-bit precision mantissas for x, y, and z. They give greater precision when displayed than the standard PS 330 vectors because the concatenation with the transformation matrix is done in double precision. They are provided because they can be used when a faster update rate is required with greater precision. They are approximately 1.5 times as fast as a Vec3bd.

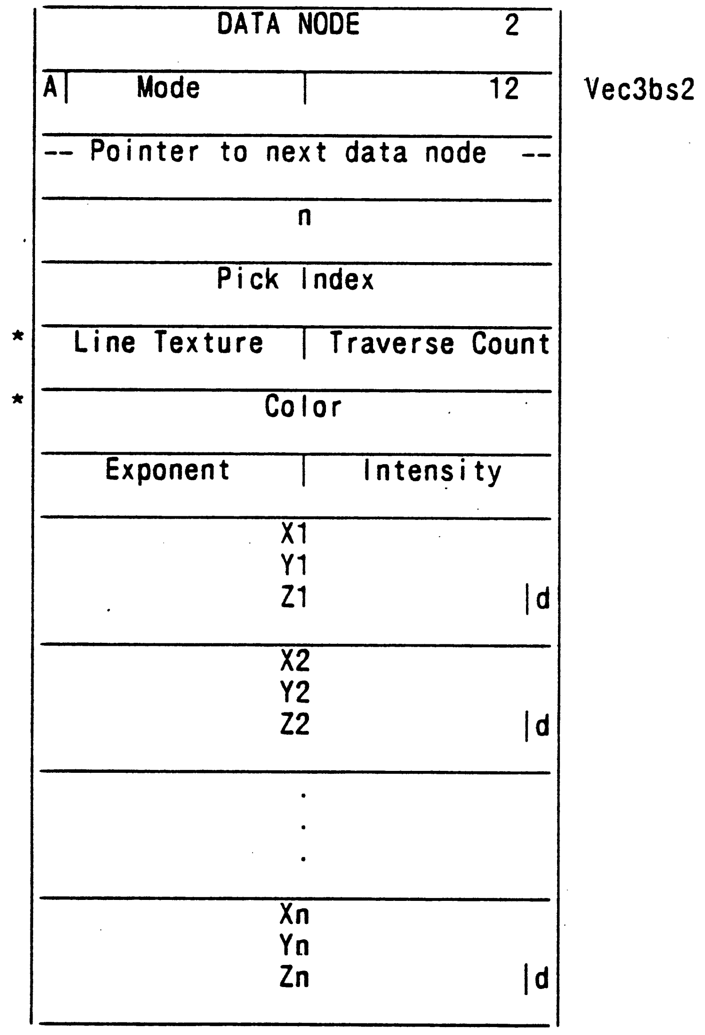


Figure 5-19. Block-Normalized Data Node - 3D (Vec3bs2)

Single precision vectors, double precision matrix multiply.

Vec2bs2

Vec2bs2 are 2D vectors, block-normalized, with 16-bit precision mantissas for x, and y. They give greater precision when displayed than the standard PS 330 vectors because the concatenation with the transformation matrix is done in double precision. They are provided because they can be used when a faster update rate is required with greater precision.

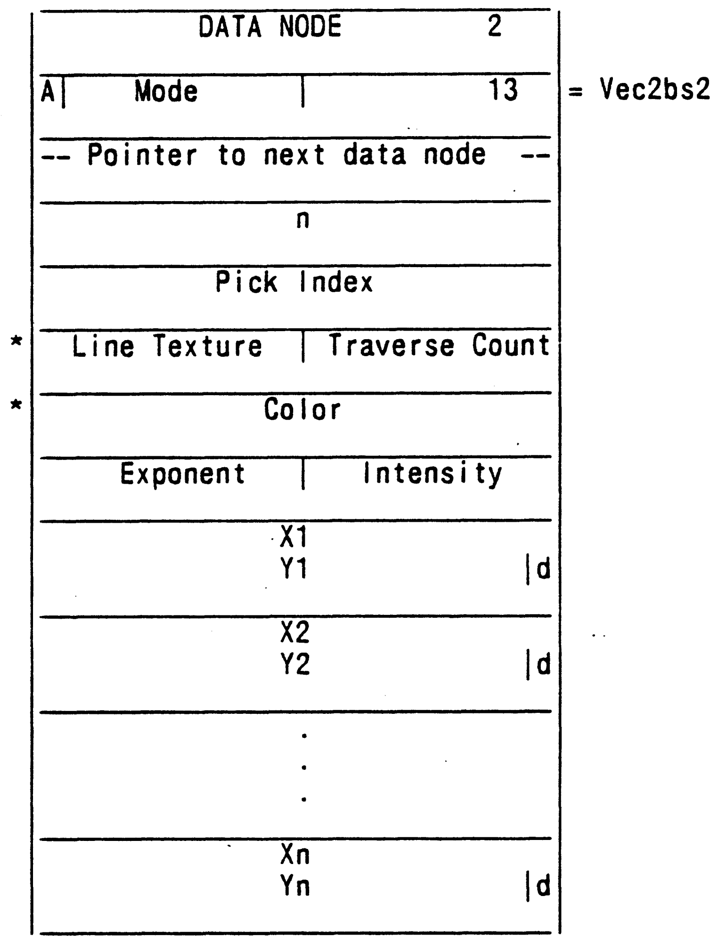


Figure 5-20. Block-Normalized Data Node - 2D (Vec2bs2)

Single precision vector, double precision matrix multiply.

DstringD

DstringD characters are similar to the standard PS 330 characters; however, arithmetic used in positioning characters is performed in double precision.

```
  |-----------------------------------|
  |          DATA NODE            2   |
  |-----------------------------------|
  |A|    Mode       |             6   |  DstringD
  |-----------------------------------|
  |-- Pointer to next data node    -- |
  |-----------------------------------|
  |     Number of Characters          |
  |-----------------------------------|
  |          Pick Index               |
  |-----------------------------------|
* | Line Texture  | Traverse Count    |
  |-----------------------------------|
* |             Color                 |
  |-----------------------------------|
  |--           Tx(H)              -- | -
  |             Tx(L)                 | |
  |-----------------------------------| |
  |--           Ty(H)              -- | > character
  |             Ty(L)                 | |  translation
  |-----------------------------------| |
  |--           Tz(H)              -- | |
  |             Tz(L)                 | -
  |-----------------------------------|
* |   Exponent    |///////////////////|
  |-----------------------------------|
* |           Exponent                | -
  |-----------------------------------| |
* |            M[1,1]                 | |
  |-----------------------------------| |
* |            M[1,2]                 | > 2 x 2 character
  |-----------------------------------| |  matrix
* |            M[2,1]                 | |
  |-----------------------------------| |
* |            M[2,2]                 | -
  |-----------------------------------|
  |           Delta x                 | -
  |-----------------------------------|  > Spacing between
  |           Delta y                 | -  characters
  |-----------------------------------|  (implied exponent of 8)
```

Figure 5-21. Character String Data Node (DstringD)

(continued on next page)

(continued from previous page)

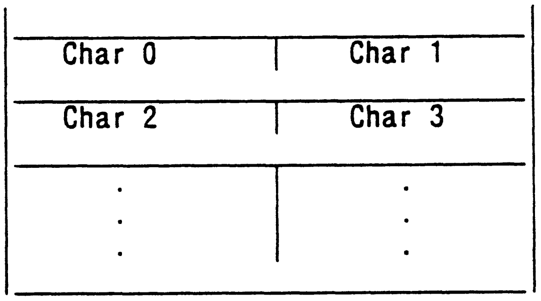


Figure 5-21. Character String Data Node (DstringD)

7-bit precision characters, double-precision multiply.

6. PS 300 ETHERNET DATA TRANSFER DESCRIPTIONS

Information in this chapter is provided for users who want to write their own host software to communicate with the PS 300 through the interface.

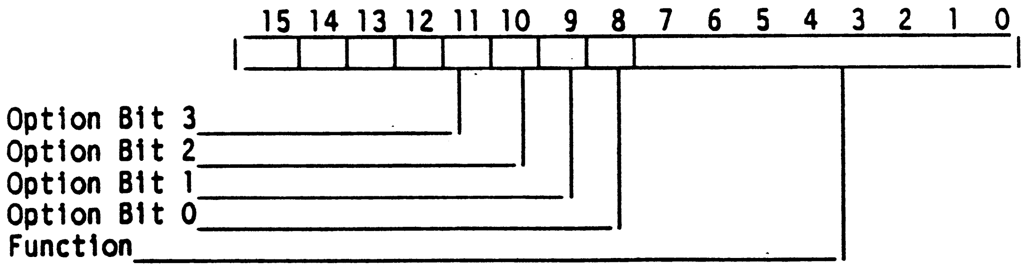
This chapter provides descriptions of the blocks of data that are used by the GPIO to transfer data between the host and locations in the PS 300, such as system functions and mass memory.

WORD DESCRIPTION

The first 4 bytes of data sent to the PS 300 are used to determine the logical or physical I/O function to perform. The bytes are used as two 16-bit words. The first word is the option and function word. The second word denotes the number of bytes in the PS Multiplex Message (this does not include the two PS Multiplex words). Note that the host computer must send the byte with the most significant bits first for both the function and count words.

When an option asks for bytes to be swapped, bytes will be swapped on 16-bit word boundaries. When an option asks for bytes to be reordered in a Control Word which is 16 bits long (number of blocks to write or Block Word Count), the bytes will be swapped. When an option asks for the bytes to be reordered in Control Word Address (32 bits long), the byte order will be reversed (bytes 3,2,1 and 0 will be reordered to 0,1,2 and 3).

PS Multiplex Function Word



FUNCTIONS

X'01' - Logical Write (from host to PS 300).
Option Bit 1 Set - Swap Bytes in Data Words.
Option Bit 2 Set - Swap Bytes in Data Words of next Logical Read.

X'02' - Logical Read (from PS 300 to host).
Option Bit 1 Set - Data Words have bytes swapped.

X'03' - Physical Write (from host to PS 300 Mass Memory).
Option Bit 0 Set - Reorder Bytes in Control Words.
Option Bit 1 Set - Swap Bytes in Data Words.
Option Bit 2 Set - Swap bytes in Frame Count.
Option Bit 3 Set - Send Frame Count Reply when write is complete.

X'05' - Physical Read Request (from host to GPIO).
Option Bit 0 Set - Reorder Bytes in Control Words.
Option Bit 2 Set - Swap Bytes in Data Words of Physical Read Reply.

X'06' - Physical Read Reply (from PS 300 Mass Memory to host).
Option Bit 0 Set - Control Words have bytes reordered.
Option Bit 1 Set - Data Words have bytes swapped.

X'07' - Synchronous Physical Write (from host to PS 300 Mass Memory).
Option Bit 0 Set - Reorder Bytes in Control Words.
Option Bit 1 Set - Swap Bytes in Data Words.
Option Bit 2 Set - Swap bytes in Frame Count.
Option Bit 3 Set - Send Frame Count Reply when write is complete.

X'09' - Lookup Name Entity Request (from host to PS 300).
Option Bit 2 Set - Reorder Bytes in Address.

X'0A' - Lookup Name Entity Reply (from PS 300 to host).
Option Bit 1 Set - Address has bytes reordered.

X'0B' - Frame Count Request (from host to PS 300).
Option Bit 2 Set - Swap bytes in Frame Count Reply.

X'0C' - Frame Count Reply (from PS 300 to host).
Option Bit 1 Set - Bytes in Frame Count have been swapped.

X'81' - Diagnostic Loopback Write (from host to GPIO).

X'82' - Diagnostic Loopback Read (from GPIO to host).

X'83' - GPIO Statistics Request (from host to GPIO).
Option Bit 2 Set - Swap Bytes in Statistical Reply Data.
Option Bit 3 Set - Reset Statistics after they are read.

X'84' - GPIO Statistics Reply (from GPIO to host).
Option Bit 1 Set - Data Words have bytes swapped.

Logical Write

Logical Write is used to transfer data between the host and the PS 300 system function network. Up to 32K bytes can be transferred during one Logical Write.

Option Bits	X'01'
Message Byte Count	
Data Up to 32K bytes	

The above table is 16 bits wide. Data may contain an odd number of bytes. If Option Bit 1 is set, the GPIO will swap bytes on 16-bit word boundaries before sending the data to the PS 300. If Option Bit 2 is set, the GPIO will swap bytes of the data in the next Logical Read.

(Revision A1)

Logical Read

Logical Read is used to transfer data between the PS 300 system function network and the host. Up to 32K bytes can be transferred during one Logical Read.

Option Bits	X'02'
Message Byte Count	
Data Up to 32K bytes	

The above table is 16 bits wide. Data may contain an odd number of bytes. If Option Bit 1 is set, the GPIO has swapped bytes on 16 bit word boundaries before sending the data to the host.

Physical Write

Physical Write is used to transfer data between the host and PS 300 mass memory. The maximum size of a Physical Write excluding the 2 Physical I/O Multiplexing Words is 32K bytes.

```
 | ______________________________ |
 | Option Bits |     X'03'        |
 | ______________________________ |
 |    Message Byte Count          |
 | ______________________________ |
 | Number of Blocks to Write      |
 | ______________________________ |
 |                                |
 - Block  1 Destination Addr.     -
 |                                |
 | ______________________________ |
 |    Block  1 word count.        |
 | ______________________________ |
 | Block  1 first data word       |
              ...
              ...
 | Block  1 last data word        |
 | ______________________________ |
 |                                |
 - Block  2 Destination Addr.     -
 |                                |
 | ______________________________ |
              ...
              ...
 | Block  n last data word        |
 | ______________________________ |
```

The table is 16 bits wide. If Option Bit 0 is set, the GPIO will reorder bytes in the control words (number of blocks to write, block destination addresses, and block word counts) before using them. If Option Bit 1 is set, the GPIO will swap bytes in the data words before writing them to mass memory.

If Option Bit 3 is set, the GPIO will send the current PS 300 Frame Count (Frame Count Reply) to the host after the write is complete. If Option Bit 2 is set, the Frame Count bytes will be reversed in the Frame Count Reply.

Physical Read Request

Physical Read Request is used to ask for the transfer of data from the PS 300 mass memory back to the host. A Physical Read Reply will be used to transmit the data.

```
| Option Bits |     X'05'      |
|------------------------------|
|      Message Byte Count      |
|------------------------------|
| Number of Blocks to Write    |
|------------------------------|
-    Block  1 Source Addr.     -
|------------------------------|
|    Block  1 word count.      |
|------------------------------|
-    Block  2 Source Addr.     -
|______________________________|
               ...
               ...
|    Block  n word count       |
|______________________________|
```

The table is 16 bits wide. If Option Bit 0 is set, the GPIO will reorder bytes in the control words (number of blocks to write, block destination addresses, and block word counts) before using them. Also, if Option Bit 0 is set, the control words in the Physical Read Reply will be reordered. If Option Bit 2 is set, the Physical Read Reply will swap data words on 16-bit word boundaries before sending to the host. The maximum size of a Physical Read Request excluding the Physical I/O Multiplex Words is 1024 Bytes. The total size of a Physical Read Reply generated by a request is 32K bytes (excluding the PS Multiplexing Words).

Physical Read Reply

Physical Read Reply is the reply to a Physical Read Request.

```
 _____________________________
| Option Bits |      X'06'    |
|_____________________________|
|     Message Byte Count      |
|_____________________________|
| Number of Blocks to Write   |
|_____________________________|
|                             |
-   Block  1 Source Addr.     -
|                             |
|_____________________________|
|     Block  1 word count.    |
|_____________________________|
| Block  1 first data word    |
               ...
               ...
| Block  1 last data word     |
|_____________________________|
|                             |
-    Block  2 Source Addr.    -
|                             |
|_____________________________|
               ...
               ...
| Block  n last data word     |
|_____________________________|
```

The above table is 16 bits wide. If Option Bit 0 is set, the GPIO has reordered bytes in the control words (number of blocks to write, block destination addresses, and block word counts) before sending them. If Option Bit 1 is set, the GPIO has swapped bytes in the data words before sending them to the host.

Synchronous Physical Write

Synchronous Physical Write is used to transfer data between the host and PS 300 mass memory. The GPIO will wait until a new frame is started in the PS 300 before transferring the data. The maximum size of a Synchronous Physical Write excluding the 2 Physical I/O Multiplexing Words is 1024 Bytes.

```
 | Option Bits |      X'07'      |
 |_______________________________|
 |     Message Byte Count        |
 |_______________________________|
 | Number of Blocks to Write     |
 |_______________________________|
 |                               |
 - Block  1 Destination Addr.    -
 |                               |
 |_______________________________|
 |      Block  1 word count.     |
 |_______________________________|
 | Block  1 first data word      |
              ...
              ...
 | Block  1 last data word       |
 |_______________________________|
 |                               |
 - Block  2 Destination Addr.    -
 |                               |
 |_______________________________|
              ...
              ...
 | Block  n last data word       |
 |_______________________________|
```

The above table is 16 bits wide. If Option Bit 0 is set, the GPIO will reorder bytes in the control words (number of blocks to write, block destination, addresses, and block word counts) before using them. If Option Bit 1 is set, the GPIO will swap bytes in the data words before writing them to mass memory.

If Option Bit 3 is set, the GPIO will send the current PS 300 Frame Count (Frame Count Reply) to the host after the write is complete. If Option Bit 2 is set, the Frame Count bytes will be reversed in the Frame Count Reply.

Lookup Named Entity Request

The Lookup Named Entity Request is used to request the GCP to find an Entity Name in memory and return the address of the Entity Name by a Lookup Named Entity Reply.

Option Bits	X'09'
Message Byte Count	
Entity Name Up to 138 bytes	

The above table is 16 bits wide. Data may contain an odd number of bytes. If Option Bit 2 is set, the GPIO will reorder the bytes of the returned address in the generated Lookup Named Entity Reply.

Lookup Named Entity Reply

Lookup Named Entity Reply is a reply to a Lookup Named Entity Request.

Option Bits	X'0A'
Message Byte Count	
Entity Name Address	

The above table is 16 bits wide. If Option Bit 1 is set, the GPIO has reordered the bytes of the address before sending it to the host.

Frame Count Request

The Frame Count Request is used to request the GPIO to return the current PS 300 Frame Count by a Frame Count Reply.

Option Bits	X'0B'
0	

The above table is 16 bits wide. If Option Bit 2 is set, the Frame Count bytes will be swapped in the Frame Count Reply.

Frame Count Reply

Frame Count Reply is a reply to a Frame Count Request.

Option Bits	X'0C'
2	
Frame Count	

The above table is 16 bits wide. If Option Bit 1 is set, the GPIO has swapped the bytes of the Frame Count.

Diagnostic Loopback Write

The Diagnostic Loopback Write is used to transfer data from the host to the GPIO and to generate a Diagnostic Loopback Read with the same data.

Option Bits	X'81'
Message Byte Count	
Data Up to 512 bytes	

The above table is 16 bits wide. Data may contain an odd number of bytes.

Diagnostic Loopback Read

The Diagnostic Loopback Read will send data of a previous Diagnostic Loopback Write back to the host.

Option Bits	X'82'
Message Byte Count	
Data Up to 512 bytes	

The above table is 16 bits wide. Data may contain an odd number of bytes.

GPIO Statistics Request

The GPIO Statistics Request is used to generate a GPIO Statistics Reply containing various statistics from the GPIO Board.

Option Bits	X'83'
0	

The above table is 16 bits wide. If Option Bit 2 is set, the GPIO will swap bytes of the data on the GPIO Statistics Reply. If Option Bit 3 is set, the statistics will reset after they are read.

GPIO Statistics Reply

The GPIO Statistics Reply sends various statistical information from the GPIO back to the host.

Option Bits	X'84'
46	
Data 46 bytes	

The above table is 16 bits wide. Option Bit 1 will be set if the bytes have been swapped in the data. The information returned (16-bit words) is as follows:

(Revision A1)

Ethernet Transmit and Receive Counters

Word 0	- Transmit Count Word 1 (MSB)
Word 1	- Transmit Count Word 0 (LSB)
Word 2	- Receive Count Word 1 (MSB)
Word 3	- Receive Count Word 0 (LSB)

Ethernet Level Errors

Word 4	- Number of Transmit Underflows
Word 5	- Number of Transmit Collisions
Word 6	- Number of 16 in a row Transmission Attempts
Word 7	- Number of Transmit Status Timeouts
Word 8	- Number of Receive Overflows
Word 9	- Number of Receive CRC Errors
Word 10	- Number of Receive Dribble Errors
Word 11	- Number of Receive Short Frames

NSP Errors

Word 12	- spare
Word 13	- spare
Word 14	- spare
Word 15	- Number of NSP Connection Timeouts
Word 16	- Number of Retransmissions to Host

PS Multiple Level Errors and Counters

Word 17	- Number of Unknown PS Multiplex Level Messages
Word 18	- Number of Invalid PS Multiplex Level Messages
Word 19	- Number of NSP retransmissions from host because of an exhausted resource at the PS Multiplex Level
Word 20	- Lowest Count in Logical I/O Empty Queue
Word 21	- Highest Count in Logical I/O Full Queue
Word 22	- The version number of the Microcode (2 ASCII Characters)

APPENDIX A. PROGRAM PSNODE INSTRUCTIONS

The program PSNODE can be used to convert the PS 300 DECnet node number from Area-Node format into the format required by the PS 300 SITE.DAT file. The following instructions build PSNODE (boldface shows what you must type):

```
$ FOR PSNODE.FOR
$ LINK PSNODE
```

The following is a sample run of PSNODE:

```
$ RUN PSNODE
DECnet Node Number: 1.46
The command for the SITE.DATE file is:
Send '042E' to <1>pi_o1$;
FORTRAN STOP
```

Note that JCP A2.V02 users must send address to **<1>ei_o1$;**

E&S CUSTOMER SERVICE TELEPHONE INFORMATION LIST

Evans & Sutherland Customer Engineering provides a central service numbered staffed by CE representatives who are available to take requests from 9:00 a.m. Eastern Time to 5:00 p.m. Pacific Time (7:00 a.m. to 6:00 p.m. Mountain Time). All calls concerning customer service should be made to one of the following numbers during these hours. Before you call, please have available your customer site number and system tag number. These numbers are on the label attached to your PS 300 display or control unit.

Customers in the continental United States should call toll-free:

1 + 800 + 582-4375

Customers within Utah or outside the continental United States should call Dispatch at:

(801) 582-9412

If problems arise during product installation or you have a question that has not been answered adequately by the customer engineer or the customer service center, contact the regional manager at one of the following Customer Engineering offices:

Eastern Regional Manager
(for Eastern and Central Time Zones)
(518) 885-4639

Western Regional Manager
(for Mountain and Pacific Time Zones)
(916) 448-0355

If the regional office is unable to resolve the problem, you may want to call the appropriate department manager at corporate headquarters:

National Field Operations
(for field service issues)
(801) 582-5847, ext 4843

Software Support
(for sofware issues)
(801) 582-5847, ext 4810

Technical Support
(for hardware issues)
(801) 582-5847, ext 4868

Director of Customer Engineering
(for any unresolved problem)
(801) 582-5847, ext 4840

READER COMMENT FORM **Publication Number** ____________

Title ____________

Your comments will help us provide you with more accurate, complete, and useful documentation. After making your comments in the space below, cut and fold this form as indicated, and tape to secure (please do not staple). This form may be mailed free within the United States. Thank you for your help.

How did you use this publication?

[] General information	[] As a reference manual
[] Guide to operating instructions	[] Other ____________

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR
Technical Accuracy Is the manual technically accurate?	[]	[]	[]	[]
Completeness Does the manual contain enough information?	[]	[]	[]	[]
Readability Is the manual easy to read and understand?	[]	[]	[]	[]
Clarity Are the instructions easy to follow?	[]	[]	[]	[]
Organization Is it easy to find needed information?	[]	[]	[]	[]
Illustrations and Examples Are they clear and useful?	[]	[]	[]	[]
Physical Attractiveness What do you think of the overall appearance?	[]	[]	[]	[]

What errors did you find in the manual? (Please include page numbers)____________

Name ____________	**Street** ____________
Title ____________	**City** ____________
Department ____________	**State** ____________
Company ____________	**Zip Code** ____________

All comments and suggestions become the property of Evans & Sutherland.

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 4632 SALT LAKE CITY, UTAH

POSTAGE WILL BE PAID BY ADDRESSEE

EVANS & SUTHERLAND
580 Arapeen Drive
Salt Lake City, Utah 84108

ATTN: IAS TECHNICAL PUBLICATIONS

Fold

Cut along dotted line

RM6. INTERFACES AND OPTIONS

CONTENTS

TABLES

Section RM6
Interfaces and Options

This section summarizes the interfaces and options available for the PS 390. Multiple interfaces, switching between interfaces, and the interface configuration files are also described. (Users manuals supplied with each interface contain detailed customer installation requirements and operating instructions.)

1. Interfaces

One of the most important considerations in setting up the configuration characteristics of a PS 390 graphics system is the interface between the host computer system and the PS 390. The standard data communication interface to the PS 390 is an asynchronous serial line. Several optional interfaces are available for the PS 390.

1.1 Asynchronous

Under PS 390 graphic system protocol, EIA RS-232-C is the standard interface used for serial asynchronous communication. With the exception of interface cabling and connectors, no additional hardware is required to interface the host with the PS 390. For a discussion of RS-232-C specifications and PS 390 asynchronous communication protocols, refer to Section *RM5*.

1.2 Parallel

The following optional interfaces are also available but may require additional interface hardware on the host and the PS 390.

The PS 390/UNIBUS™ Parallel Interface supports high-speed data transfers to and from a DEC/VAX™ host computer running the VMS™ operating system at 3.2 or higher.

The parallel interface uses the normal command processing mechanism in the PS 390 to construct graphic data structures and establish local action operations. When integrated with the PS 390 Graphics Support Routines, the interface provides an even greater increase in data throughput. It is especially useful in applications requiring a close coupling with the host computer.

1.3 Ethernet

The PS 390/Ethernet™ (DECNET™) Interface is a high-speed communications interface connecting a PS 390 graphics system to a DEC/VAX™ or MicroVAX™ host computer with a VMS™ operating system 3.2 or higher.

The PS 390/Ethernet (TCP/IP) Interface is a high-speed communications interface designed to connect a PS 390 graphics system to a DEC/VAX host computer running under UNIX™ BSD 4.2 or higher.

The Ethernet interfaces allow a PS 390 to link to an Ethernet data communications network. They are intended for use in office automation and distributed data processing environments to allow a selected group of computers to communicate with each other.

1.4 IBM 3278

The PS 390/IBM™ 3278 Interface allows a PS 390 graphics system to be connected to an IBM host using an IBM 3274 channel control unit to provide high-performance graphics functions while attached in the same manner as the 3278 terminal. The PS 390 supports an IBM terminal emulator when configured with this interface option. All the basic functions of the 3278 are fully supported, including basic attribute byte and keyboard functions.

1.5 IBM 5080

The PS 390/IBM™ 5088 Interface provides a high-speed, channel connect attachment between a PS 390 graphics system and an IBM host computer via an IBM 5088 controller.

The PS 390/IBM™ 5088/V.35 Interface provides remote attachment by connecting the PS 390 to a V.35 broadband modem that is attached to the IBM 5088 controller.

Both interfaces support the 5080 Capability option. This firmware option allows the user to perform most IBM 5080 operations and to run programs from the PS 390 that were written specifically for the IBM 5080, such as CATIA™ and CADAM™.

These interfaces allow the PS 390 to be connected to any IBM host computer using a standard IBM 5088 channel control unit. The PS 390 can be configured with other IBM 5080 graphics terminals on the same IBM 5088 channel control unit.

2. Multiple GPIO Interfaces

The PS 390 runtime firmware supports up to two GPIO interfaces of differing types as well as asynchronous communications installed in the same system. You received two firmware diskettes with your system: a runtime system diskette preconfigured for your site with interface communication defaults and an interface diskette for modifying system configuration. By renaming files on the diskettes you can change your default to configure a different interface when the system is booted. This is explained in section 2.1.

It is also possible to change the configuration without rebooting the PS 390 because the runtime determines which of the interfaces are in the system and initializes them all. This is achieved through runtime identification of up to two GPIOs at the first two addresses assigned to GPIO interface cards. (Refer to section 2.1 for an example of changing interface communications protocol without rebooting.)

There are some limitations to the use of multiple GPIOs. First, there cannot be two of the same type GPIO in the same system. Second, if the IBM 3278 option is included, then only one additional GPIO may be added. The 3278 GPIO running under previous PS 300 systems is not supported under the PS 390. Table 6-1 shows the possible GPIO combinations.

Table 6-1. Possible GPIO Combinations

1st GPIO	2nd GPIO
IBM 3278 (enabled on JCP)	IBM 5080
	Parallel
	Ethernet
IBM 5080	Parallel
	Ethernet
Parallel	IBM 5080
	Ethernet
Ethernet	IBM 5080
	Parallel

2.1 Interface Configuration Files

The PS 390 runtime is distributed on two diskettes and contains more files than previous PS 300 runtime diskettes. This is to allow for the many different combinations of interfaces possible with the multiple GPIO operation.

When the PS 390 is booted, the system attempts to read the file, INTFCFG.DAT. If this file is not found, the system will boot with the default interface of asynchronous, and display the message INTFCFG.DAT NOT FOUND. To boot with a default interface in addition to asynchronous, the appropriate interface file must be renamed to INTFCFG.DAT. This can be done using the Diagnostic Disk Utility program described in Section *RM12 Diagnostic Utilities*. For example,

```
Rename ETHERNET.DAT INTFCFG.DAT
```

would rename the default interface to Ethernet so that, at boot time, the interface communications protocol for Ethernet would be configured.

The following is a list of the interface file names on the diskette and which interface each file sets up.

ASYNC.DAT	Asynchronous communications
IBM3278.DAT	IBM 3278 communications
IBM5080.DAT	IBM 5080 communications
UNIBUS.DAT	Parallel interface communications
ETHERNET.DAT	Ethernet communications (for Ethernet or DECNET)

If your system hardware supports two interfaces, you can change the interface during a session without rebooting by sending the name of the interface file to input <1> of RDCFG$. For example, the following command,

```
Send 'UNIBUS' to <1>RDCFG$;
```

would change the communications protocol to the UNIBUS Parallel interface to allow parallel communications.

Table 6-2 shows the files contained on the PS 390 diskettes which are needed for a particular interface.

Table 6-2. Required Interface Files

PS 390 File Name	Async	3278	5080	Unibus	Ethernet
mmdd390J.EXS	✓	✓	✓	✓	✓
ACPCODE2.DAT	✓	✓	✓	✓	✓
ASYNC.DAT	✓				
CHARFONT.DAT	✓	✓	✓	✓	✓
CIRCLE.DAT			✓		
CONFIG.DAT	✓	✓	✓	✓	✓
DINTCODE.DAT					✓
EINTCODE.DAT					✓
ETHERNET.DAT					✓
FCNDICTY.DAT	✓	✓	✓	✓	✓
FCNTABLE.DAT	✓	✓	✓	✓	✓
FONT5080.DAT			✓		
GPIOCODE.DAT		✓			
HMSCODE.DAT	✓	✓	✓	✓	✓
HMSCOL.DAT	✓	✓	✓	✓	✓
HMSVEC.DAT	✓	✓	✓	✓	✓
IBM3278.DAT		✓			
IBM5080.DAT			✓		
IBMASCII.DAT	✓	✓	✓	✓	✓
IBMFONT.DAT	✓	✓	✓	✓	✓
IBMKEYBD.DAT	✓	✓	✓	✓	✓
INITACP.DAT	✓	✓	✓	✓	✓
INITGPIO.DAT		✓			
LINLUT.DAT	✓	✓	✓	✓	✓
LUT.DAT	✓	✓	✓	✓	✓
MSGLIST.DAT	✓	✓	✓	✓	✓
OVERLAY2.DAT	✓	✓	✓	✓	✓
PARSECODE.DAT	✓	✓	✓	✓	✓
PARSDICT.DAT	✓	✓	✓	✓	✓
PINTCODE.DAT				✓	
SINE.DAT			✓		
THULE.DAT	✓	✓	✓	✓	✓
UNIBUS.DAT				✓	

All of the interface files assume that the keyboard used is a VT100-style keyboard. A FALSE is sent to the keyboard handler (either IBMKBD or KBHANDLER) at the end of the file. To use an IBM-style keyboard, the command in the interface file must be changed to send TRUE to the keyboard handler. For example,

```
Send True to <2>Kbhandler;
```

would accomplish this.

2.2 Ethernet/DECNET Interface

The GPIO interface hardware for Ethernet and DECNET is the same. The only difference is the microcode that is loaded into the GPIO. Therefore, both microcode files are distributed on each diskette. The runtime attempts to load a file named EINTCODE.DAT. Ethernet is the default on the diskette. The file for the DECNET interface is DINTCODE.DAT. If your system supports the DECNET interface, DINTCODE.DAT must be renamed to EINTCODE.DAT to load the DECNET microcode into the GPIO. This can be accomplished by using the Diagnostic Utility program.

NOTE

For additional information on customer hardware and software installation requirements for the various interfaces refer to the Customer Installation and User Manuals supplied by E&S.

3. System Options

3.1 Memory Card Option

Up to two 1 MByte cards can be added to expand the standard JCP resident 2 MByte of memory. The cards can be installed in the PS 390 at the factory or can be installed at the customer location.

3.2 User-Written Function Facility

The User-written Function Facility is designed to allow programmers to write and use new functions to suit individual applications and needs.

All PS 390 graphics systems include a set of intrinsic functions which allow complex graphics actions to be accomplished locally within the PS 390. These functions are the user interface between the programmer, display structures, interactive devices, and high-performance graphics facilities in the PS 390.

User-written functions expand the capabilities of the PS 390 by giving the programmer the power to create unique functions, or to combine large networks of intrinsic functions into a single function that performs all the same operations, yet is much simpler in design and operation.

A user-written function is written on the host computer as a procedure for the Motorola 68000, in Pascal or Motorola 68000 assembly language. Through the cross-compiling and linking software, the procedure is translated into S-record host files which are then transferred to the PS 390 memory. The function is identified by its user-given name and stays in memory as long as its name remains there. Once installed in the PS 390, User-Written Functions can be used in the same way as the intrinsic functions.

3.3 Advanced 3D Visualization Firmware

The Advanced 3D Visualization Firmware option allows users to create objects as polygons and to display hidden-line removed and sectioned views of polygonally-defined wireframe objects. Smooth-shaded renderings of polygonal models can be displayed that take advantage of numerous attribute settings for color, multiple light sources, specularity, transparency, and polygon edge enhancement. In addition the PS 390 can be used as a frame buffer for the display of host-generated, run length-encoded images.

RM7. HOST INPUT DATA FLOW

CONTENTS

TABLE

Section RM7
Host Input Data Flow

This section discusses host input data flow in the PS 390, and includes a description of the functions that direct data flow, the routing functions and routing bytes, and the channels that data can be routed to. Function names that appear in capital letters are *instances* of intrinsic system and user functions. The intrinsic system and user functions (also capitalized) appear with the "F:" prefix.

1. Data Reception and Routing Network

Data enters the PS 390 through one or more input functions. In systems with the asynchronous interface, an instance of F:DEPACKET (an intrinsic user function) receives host input and passes it to an instance of F:CIROUTE(n) (an intrinsic user function). There are two instances of F:CIROUTE(n), one for count mode (CIROUTE0) and one for escape mode (CIROUTE20). CIROUTE0 examines the first character it receives (the character following the count bytes in count mode or the character following the <FS> character in escape mode) to determine where the packet message is to be sent. This character is the routing byte, and is used to select the appropriate channel for the data in the PS 390. Data channels may include lines to the terminal emulator, the command interpreter, the disk writing function, the raster function, and other intrinsic functions. A base character (defined on Input <2> of CIROUTE0) is subtracted from this routing character before it is used to select the output channel. The base character defaults to the character zero ("0").

All other interfaces send host input through special interface functions which pass it to a count mode instance of F:CIROUTE(n). For the Parallel and Ethernet interfaces, the input may be routed through CIROUTE30. For the IBM 3278 and IBM 5080 interfaces, the input is routed through CIROUTE0. CIROUTE0, CIROUTE20, and CIROUTE30 are functionally identical.

The definitions for the inputs and outputs of intrinsic system functions and intrinsic user functions are described in Section *RM2*. Escape and count modes are discussed in Section *RM5*.

2. Routing Byte Definitions

The value of the routing bytes are given in the following table.

Table 7-1. Routing Byte Definitions

CIROUTE0 Output	Routing Byte	Channel Parameter	Description
1	N/A	N/A	Reserved
2	N/A	N/A	Reserved
3	0	1	Parser/Command Interpreter
4	1	2	Command Interpreter via READSTREAM
5	2	3	6-bit binary
6	3	4	Reset network for GSRs
7	4	5	Unused
8	5	6	Unused
9	6	7	Download channel for user-written functions
10	7	8	Raster
11	8	9	Polygon data
12	9	10	Unused
13	:	11	Write ASCII data to diskette
14	;	12	Close file
15	<	13	Write binary data to diskette
16	=	14	Unused
17	>	15	Channel to Terminal Emulator
18	?	16	Host message control
19	@	17	Reserved
20	A	18	Unused
21	B	19	Raster

NOTE

('?') is the HOST_MESSAGE request channel. An ASCII (1 or 2) requests a single message or multiple messages from HOST_MESSAGEB.

3. Output Port Definitions of CIROUTE0 in Count Mode

Output<1> sends out invalid routing bytes.

Output<2> sends any message that does not have a valid routing character. The message is sent to BADROUTE0 (an instance of the intrinsic user function F:CONSTANT), and the message *"Routing byte not in acceptable range"* is output as an error message to ES_TE1 (an instance of the intrinsic system function F:VT10) for screen display.

Output<3> sends messages to H_CHOP0 (an instance of the intrinsic user function F:CHOP). This function chops and parses the input command language generating proper messages for H_CI0 (an instance of the intrinsic user function F:CI). Once chopped and parsed, the message is sent on output<1> of H_CHOP0 to the Command Interpreter. H_CHOP0 is also responsible for generating syntax error messages. ASCII commands should be sent through this output.

Output<4> sends messages to READSTREAM0 (an instance of the intrinsic system function F:READSTREAM), which converts an eight-bit stream into arbitrary messages. GSR data is sent through this output or through output <5>.

Output<5> sends messages to SIXTOEIGHT0 (an instance of the intrinsic user function F:CVT6TO8) to convert six-bit to eight-bit binary. The message is then sent to READSTREAM0. GSR data is sent through this output or through output <4>.

Output<6> sends messages to RESET_RS1 (an instance of the intrinsic user function F:RESET) and RESET_HOST_MESSAGE1 (an instance of the intrinsic user function F:CONSTANT), which causes the functions accepting GSR data to be reset to the initial state.

Output<7> is unused.

Output<8> is unused.

Output<9> sends messages to SREC_GATHER0 (an instance of the intrinsic user function F:GATHER_GENFCN), which loads user-written functions.

Output<10> sends messages to RASSTR0 (an instance of the intrinsic system function F:RASTERSTREAM), which processes pixel input using run-length encoding of data from the host.

Output<11> sends messages to HPOLYSTR0 (an instance of the intrinsic user function F:HOST_POLY), which processes polygon fill commands sent from the host.

Output<12> is unused.

Output<13> sends messages to WDA0 (an instance of the intrinsic user function F:WRITEDISK), which writes ASCII commands to the diskette.

Output<14> sends messages to WDAC0 (an instance of the intrinsic user function F:CHOP), which is used to interpret the command to close the file sent via outputs <13> and <15> to the diskette.

Output<15> sends messages to WDBC0 (an instance of the intrinsic user function F:CHOP), which is used to parse binary data that will be written to the diskette.

Output<16> is unused.

Output<17> sends messages to ES_TE1 (an instance of the intrinsic system function F:VT10), which processes input for the PS 390 display screen.

Output<18> sends messages to TRIGGER_CONVB1 (an instance of the intrinsic user function F:CHARCONVERT). TRIGGER_CONVB1 then sends messages to input <1> of HOST_MESSAGEB1 (an instance of the intrinsic user function F:HOLDMESSAGE).

Output<19> sends messages to WHO1, which sends a package with the system information back to the host. This output has been retained for compatibility. It is not used on the PS 390.

Output<20> is unused.

Output<21> sends messages to RASSTR0 (an instance of the intrinsic system function F:RASTERSTREAM), which processes pixel input using run-length encoding of data from the host. This output is the same as output <10>, and has been retained for compatibility purposes. Output <10> is the recommended output since it is controlled by the Qprompt flushing mechanism by default.

Section RM8
System Function Network

The block diagrams in this section show the data flow through the PS 390 system function network. Function names that appear in capital letters in this section are instances of intrinsic system and user functions. The intrinsic function appears with the "F:" prefix. Intrinsic function descriptions are provided in Section *RM2*.

- Figure 1 shows the initial read floppy network created in the PS 390.
- Figures 2 through 26 show the host input data flow through the system function network for a PS 390 with an RS-232 interface to a host computer.
- Figures 27 through 49 show the host input data flow through the system function network for a PS 390 with an IBM host computer.
- Figure 50 shows the host input data flow through the raster system function network for a PS 390 with a DEC host computer.
- Figure 51 shows the host input data flow through the raster system function network for a PS 390 with an IBM host computer.
- Figures 52 through 55 show the host input data flow through the DEC Parallel Interface function network.

A discussion of specific instances of functions that direct data flow in the PS 390 will be found in Section *RM7, Host Input Data Flow.*

NOTE

The diagrams in this section reflect A1 firmware functionality. We will be distributing updated diagrams in a future release.

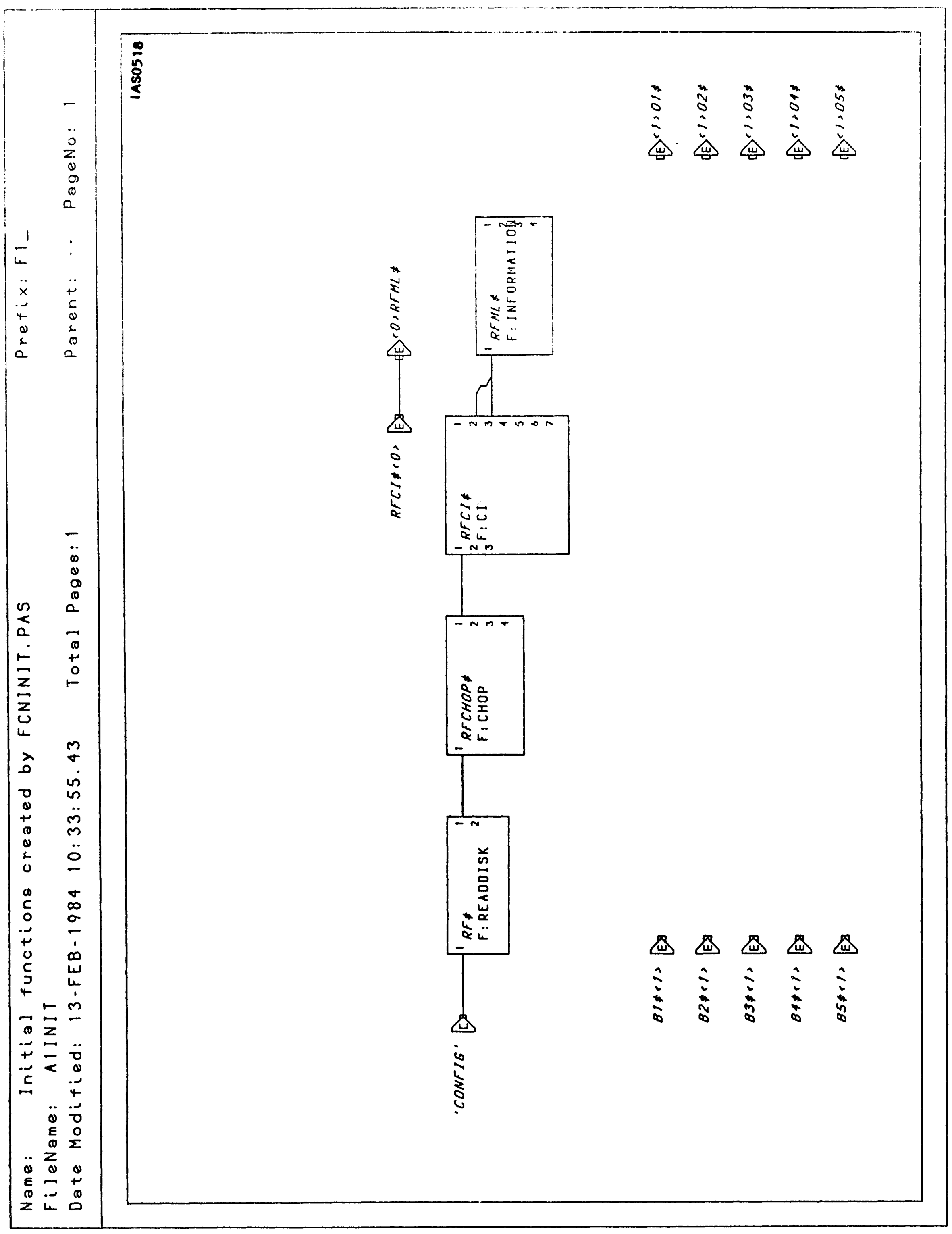


Figure 8-1. PS 390 Initial Read Floppy Network Created

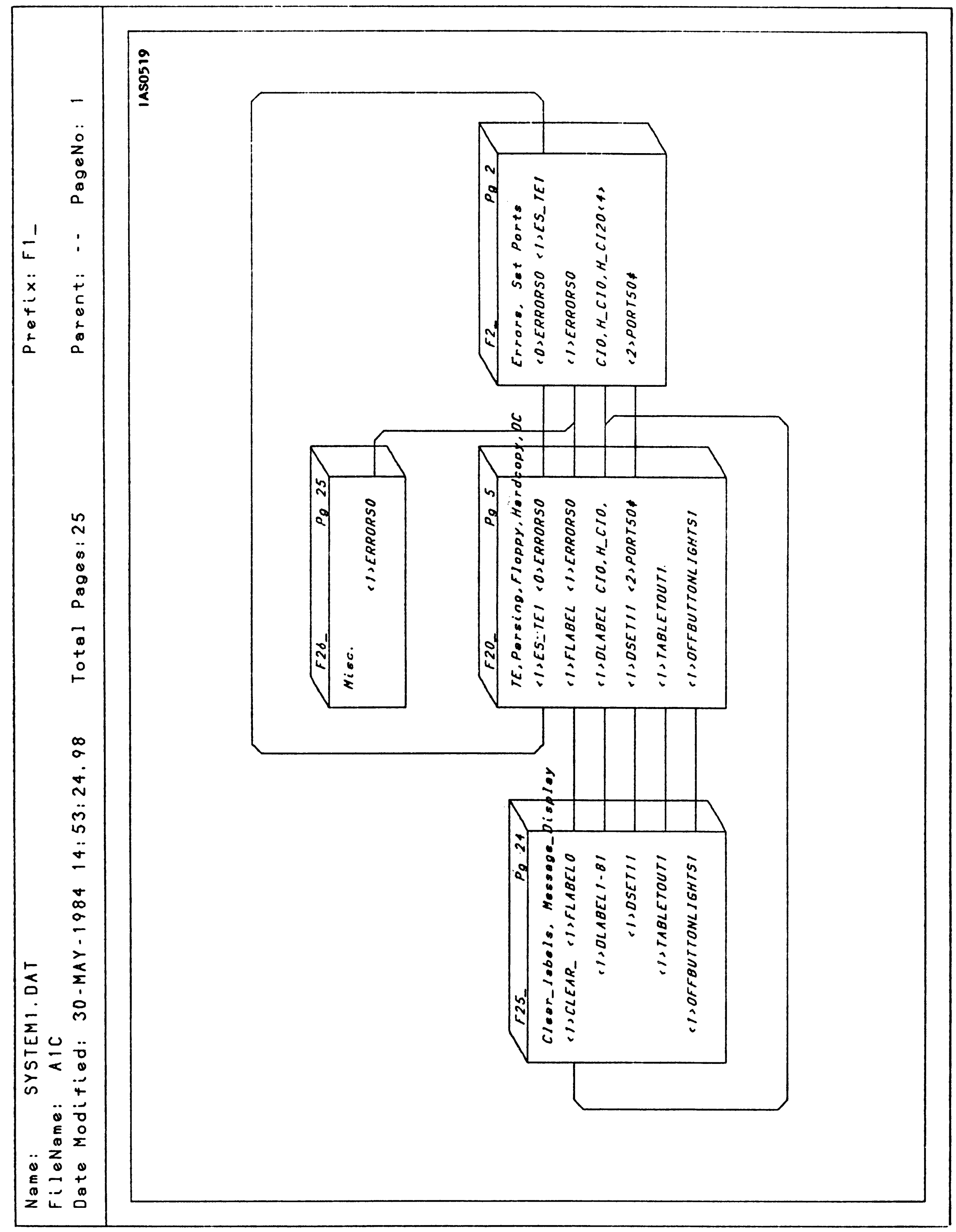


Figure 8-2. PS 390 Host Input Data Flow (RS-232 Interface)

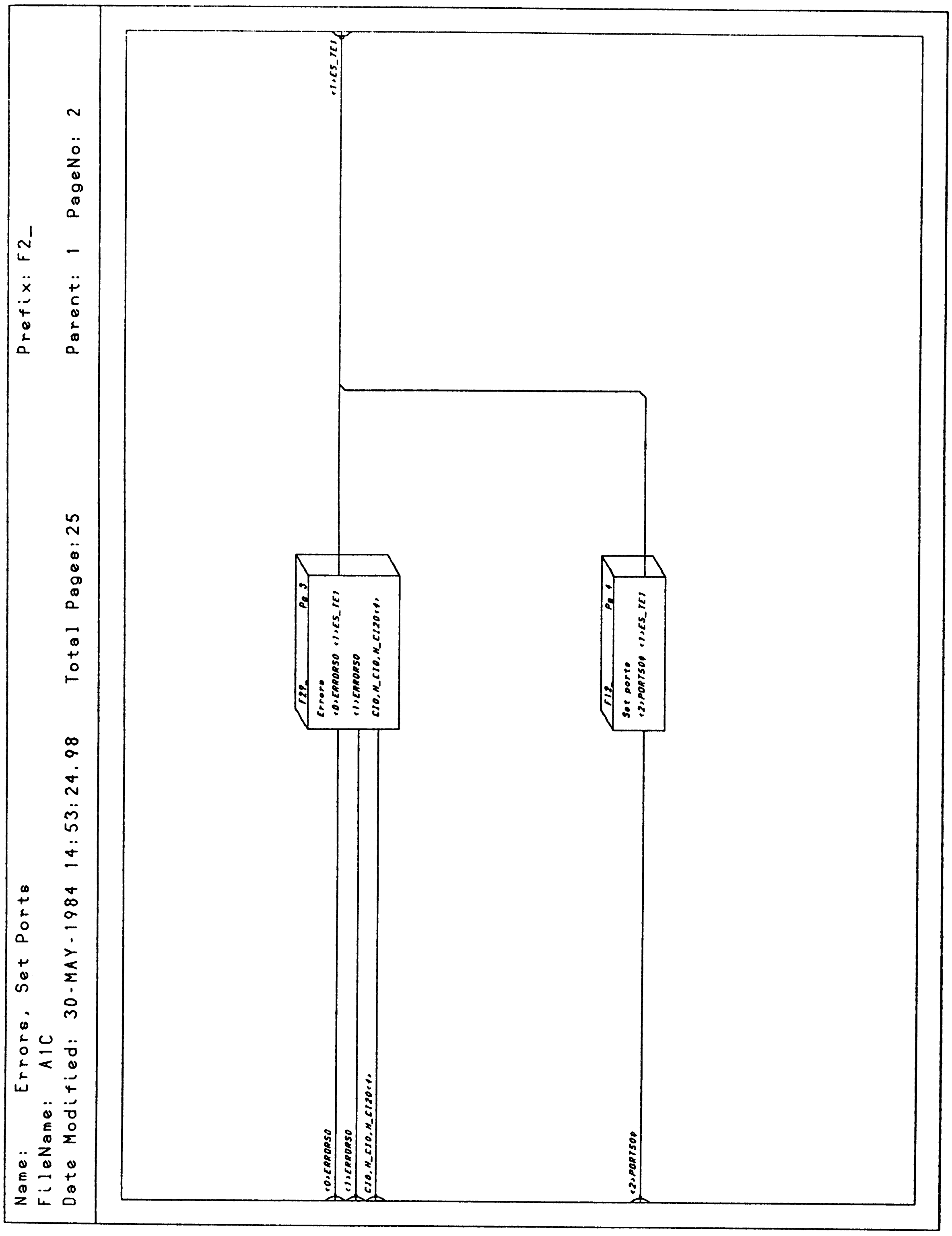


Figure 8-3. *PS 390 Host Input Data Flow (RS-232 Interface)*

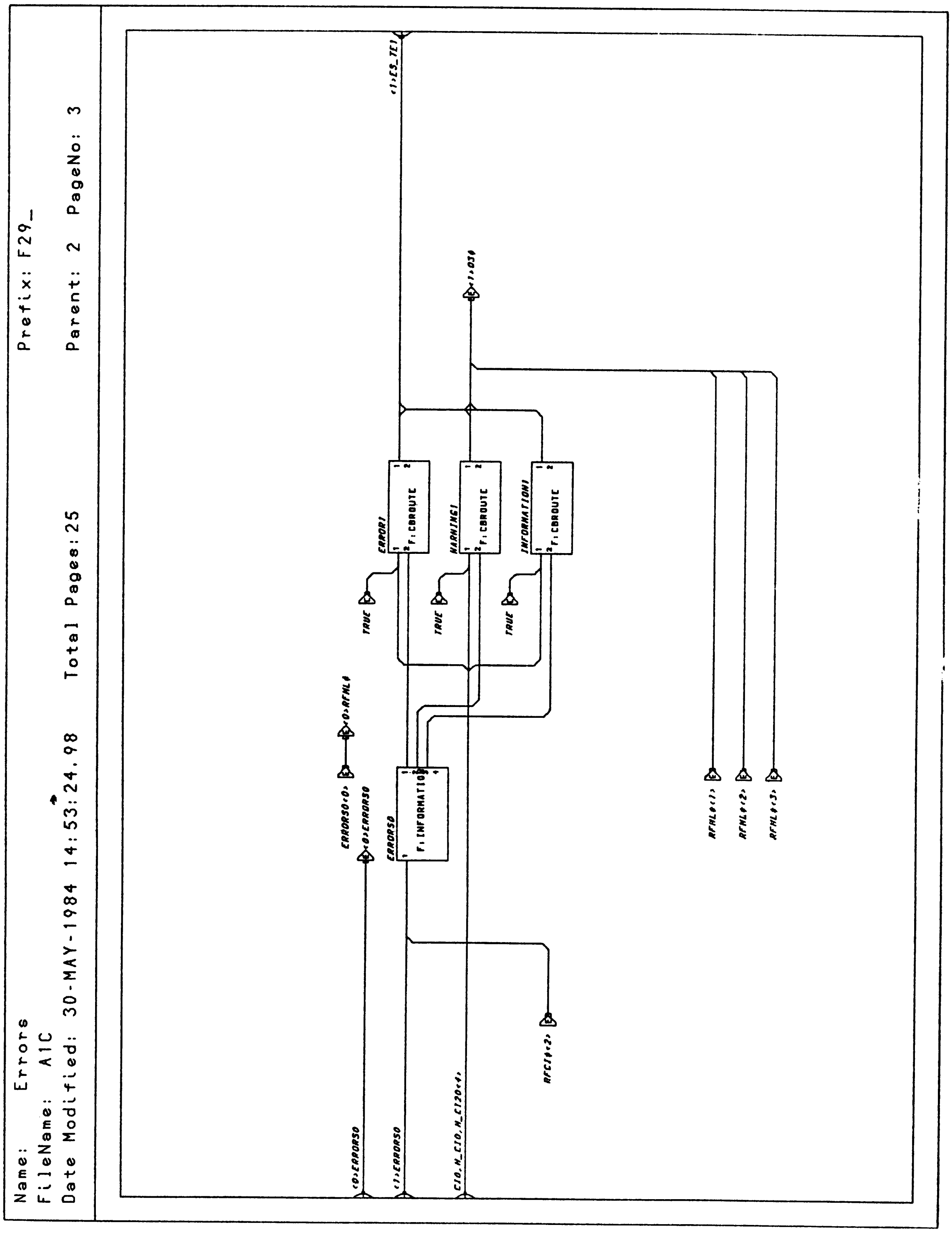


Figure 8-4. PS 390 Host Input Data Flow (RS-232 Interface)

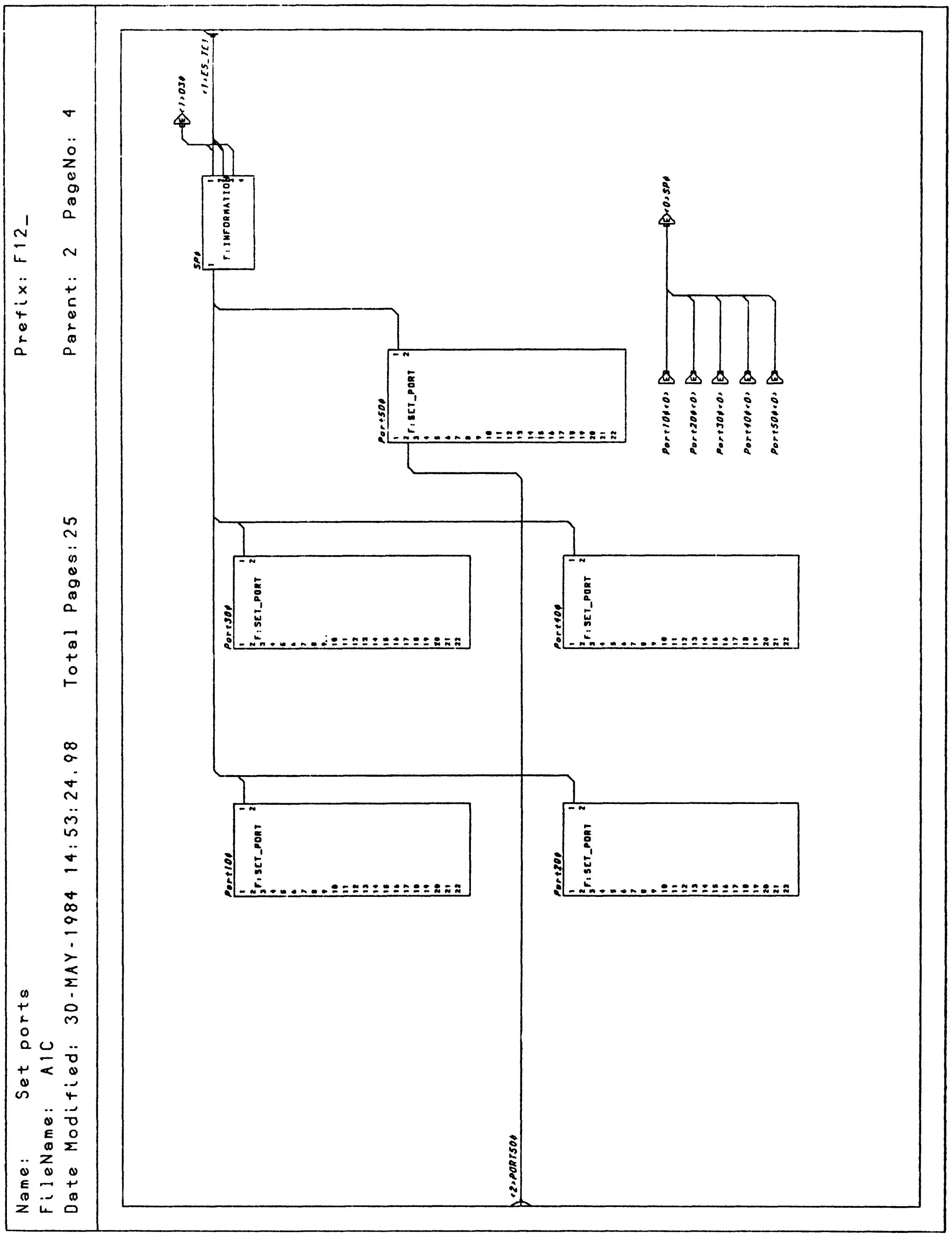


Figure 8-5. PS 390 Host Input Data Flow (RS-232 Interface)

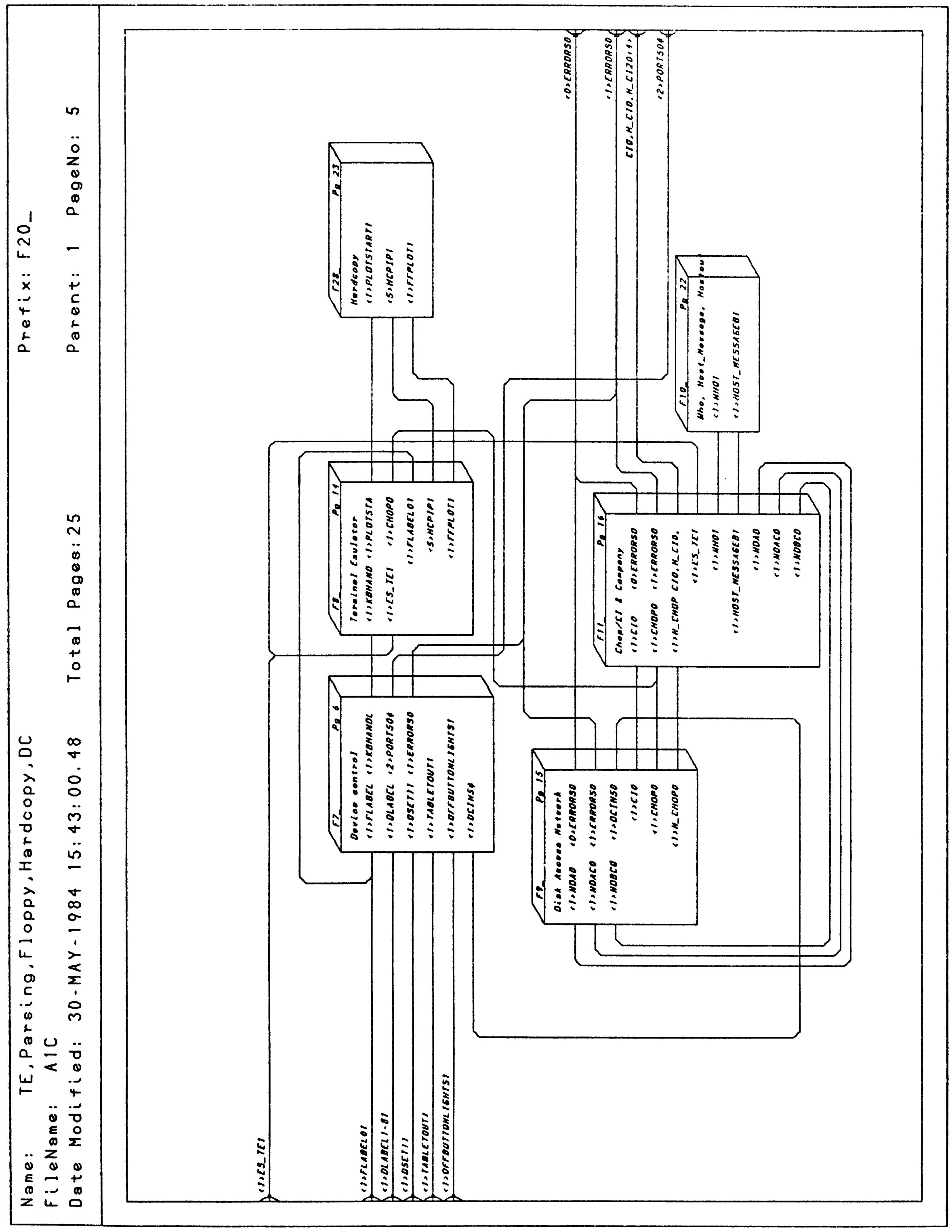


Figure 8-6. PS 390 Host Input Data Flow (RS-232 Interface)

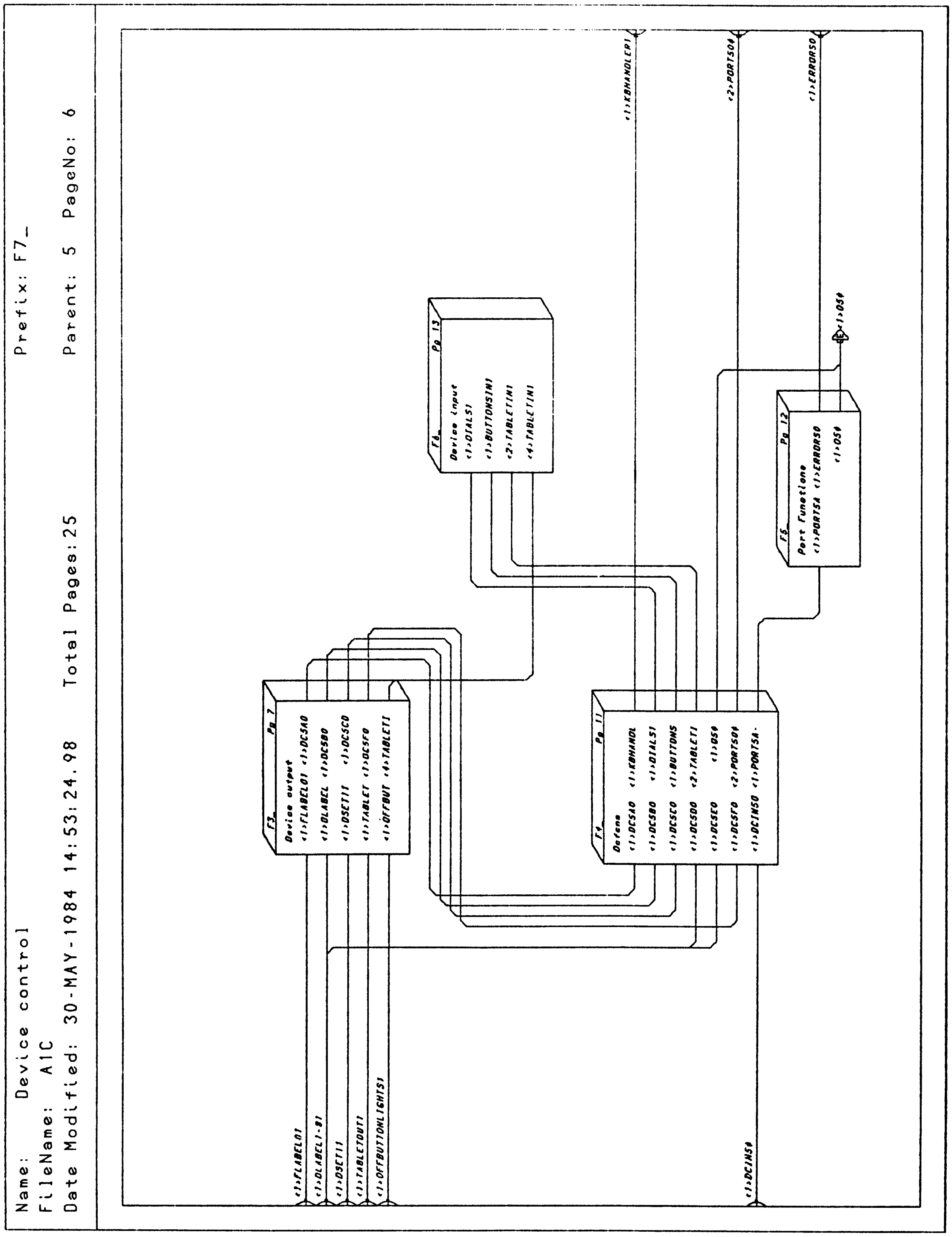


Figure 8-7. PS 390 Host Input Data Flow (RS-232 Interface)

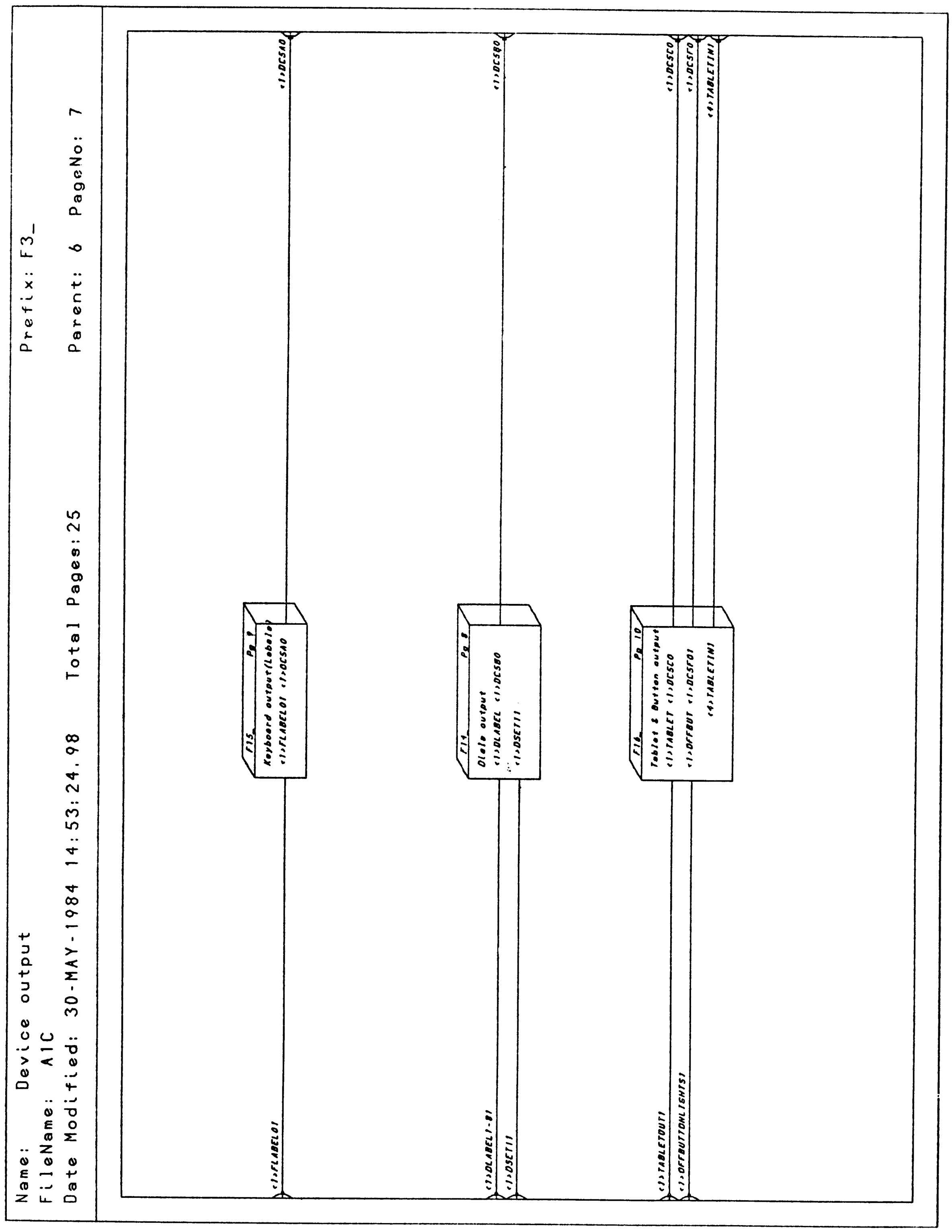


Figure 8-8. PS 390 Host Input Data Flow (RS-232 Interface)

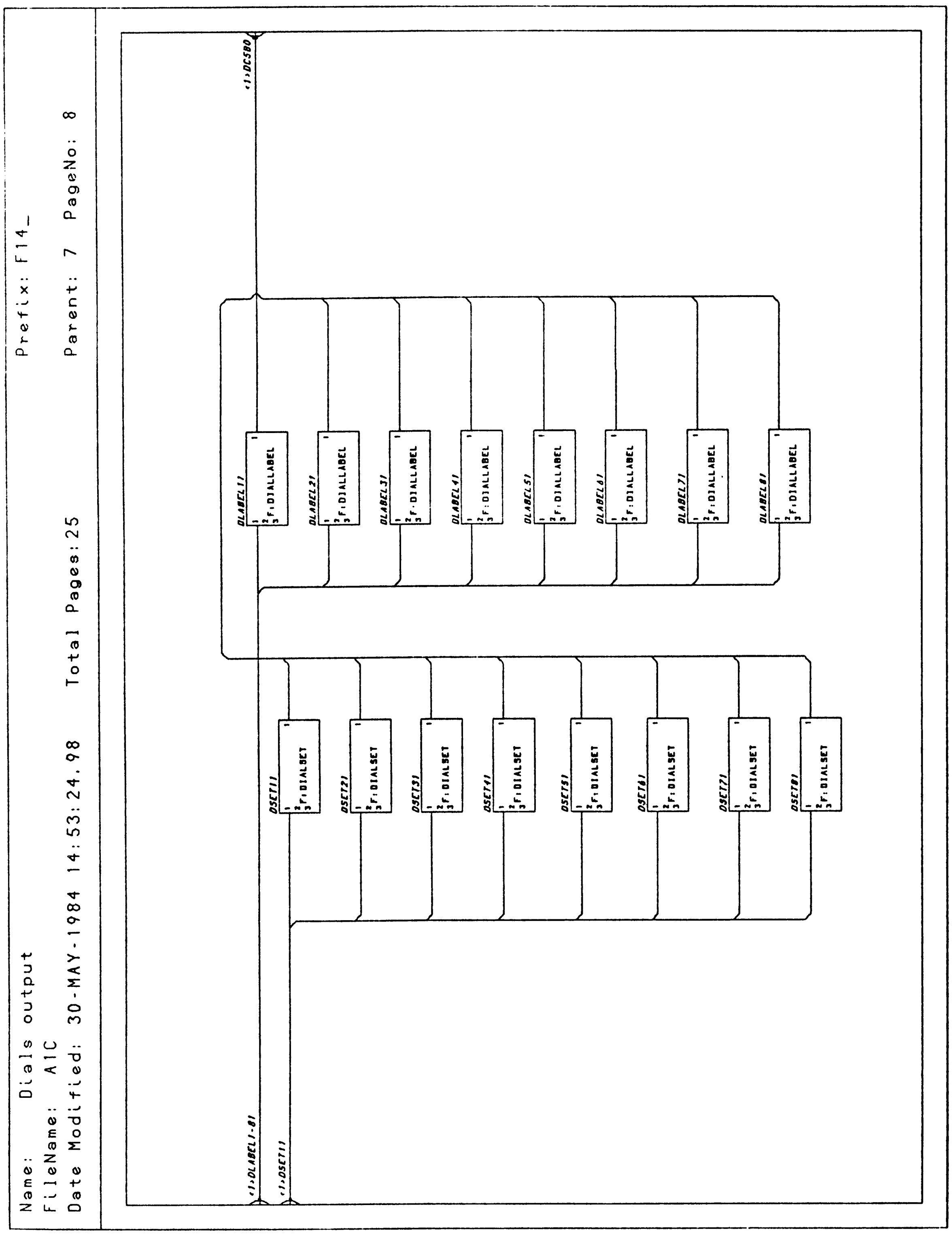


Figure 8-9. PS 390 Host Input Data Flow (RS-232 Interface)

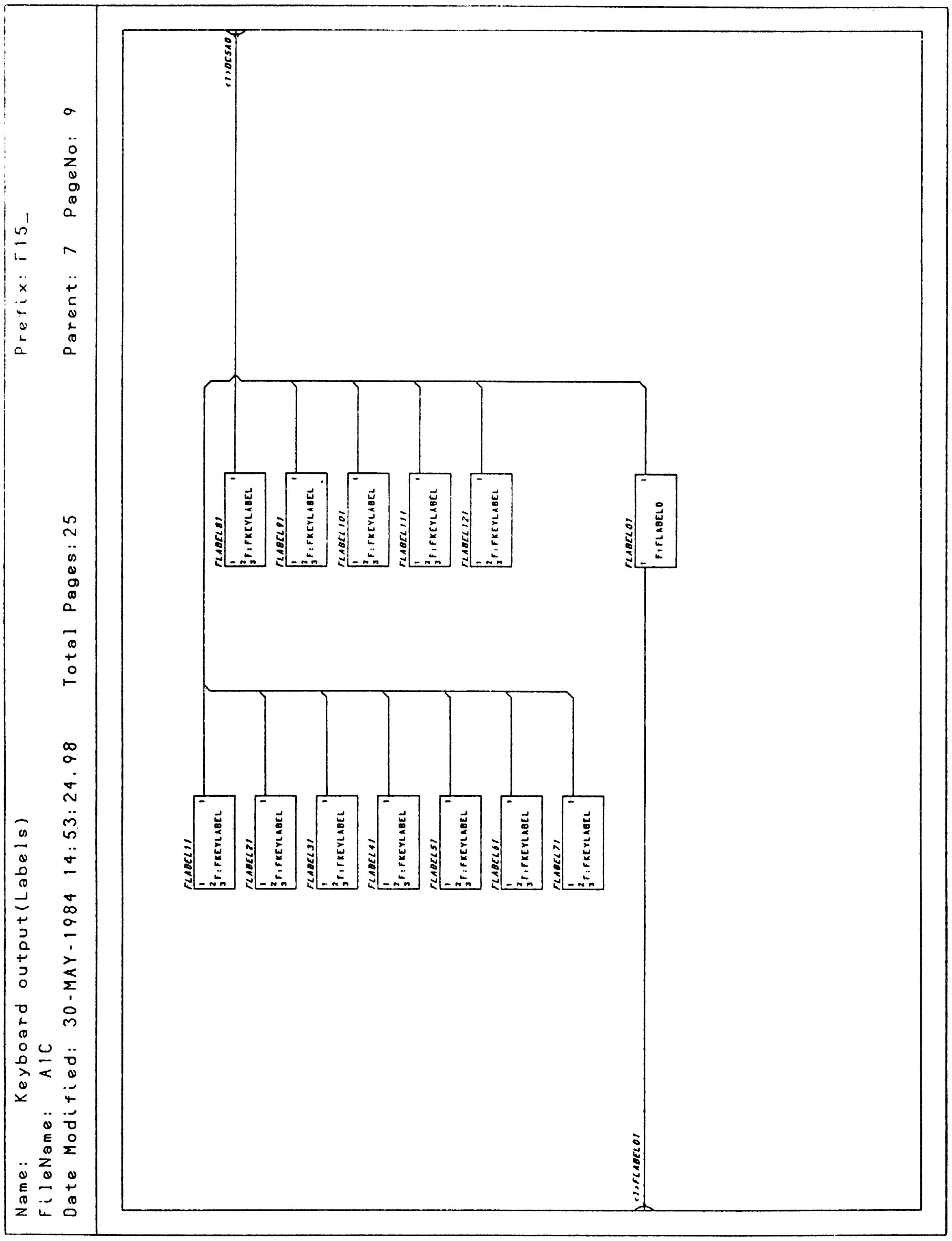


Figure 8-10. PS 390 Host Input Data Flow (RS-232 Interface)

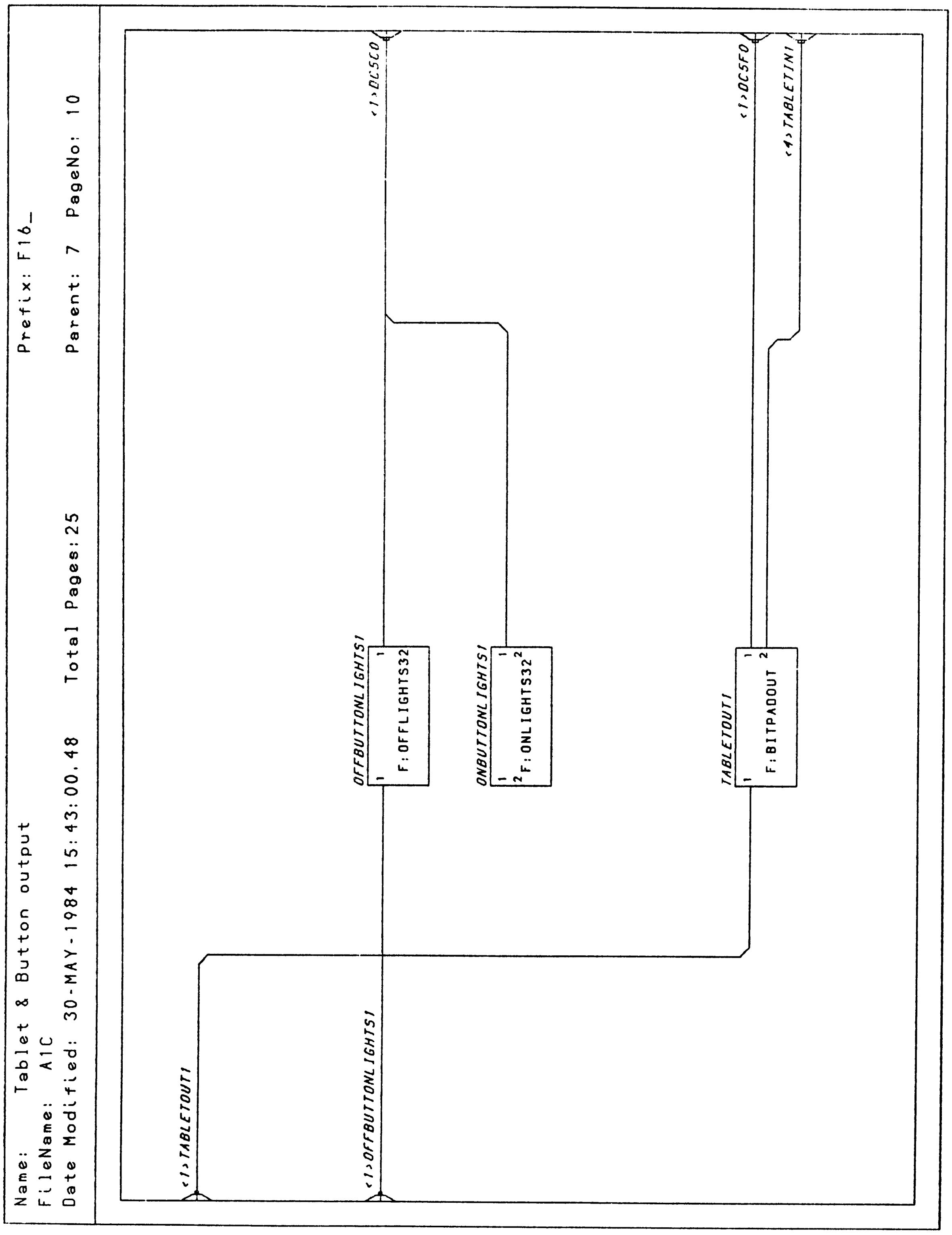


Figure 8-11. PS 390 Host Input Data Flow (RS-232 Interface)

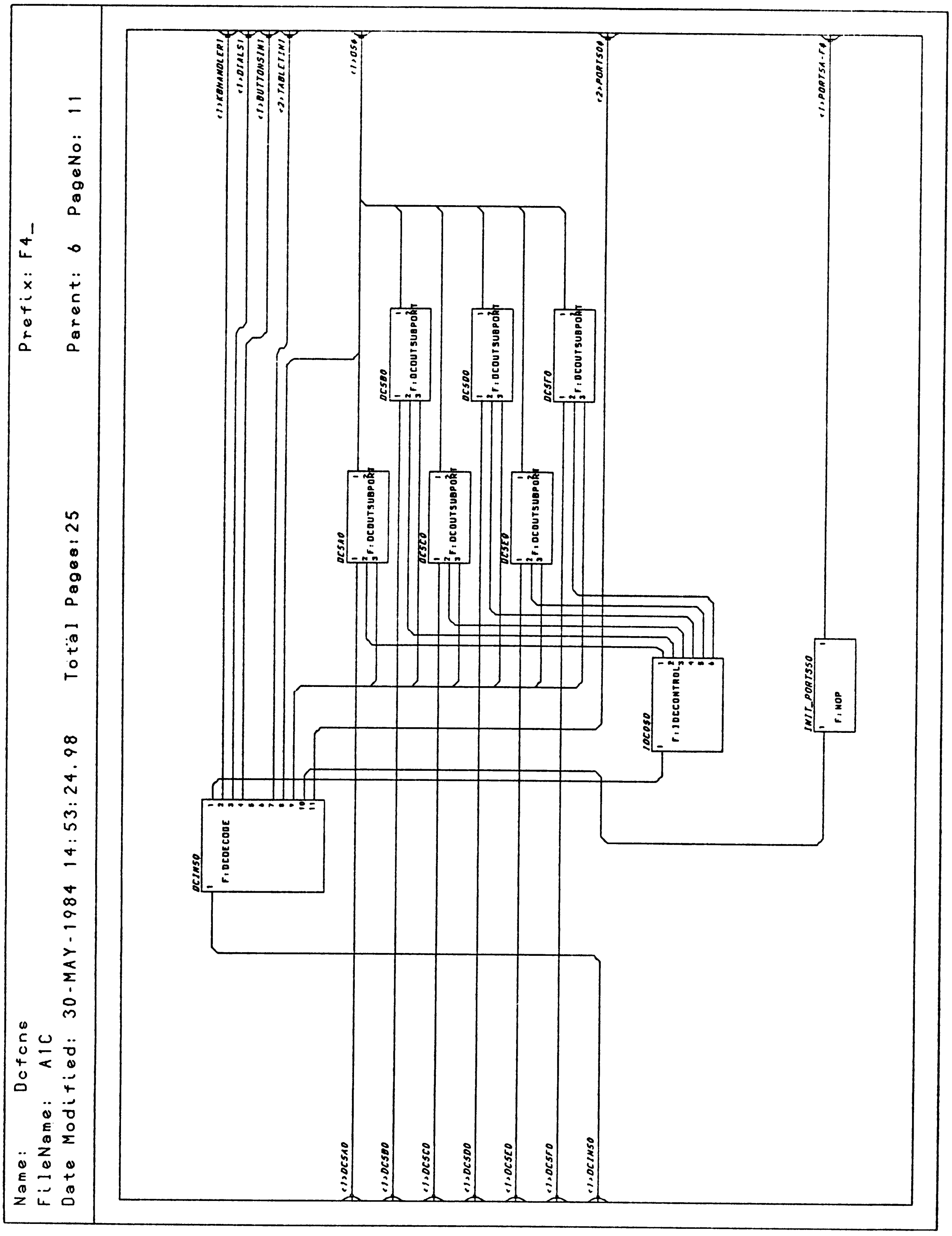


Figure 8-12. PS 390 Host Input Data Flow (RS-232 Interface)

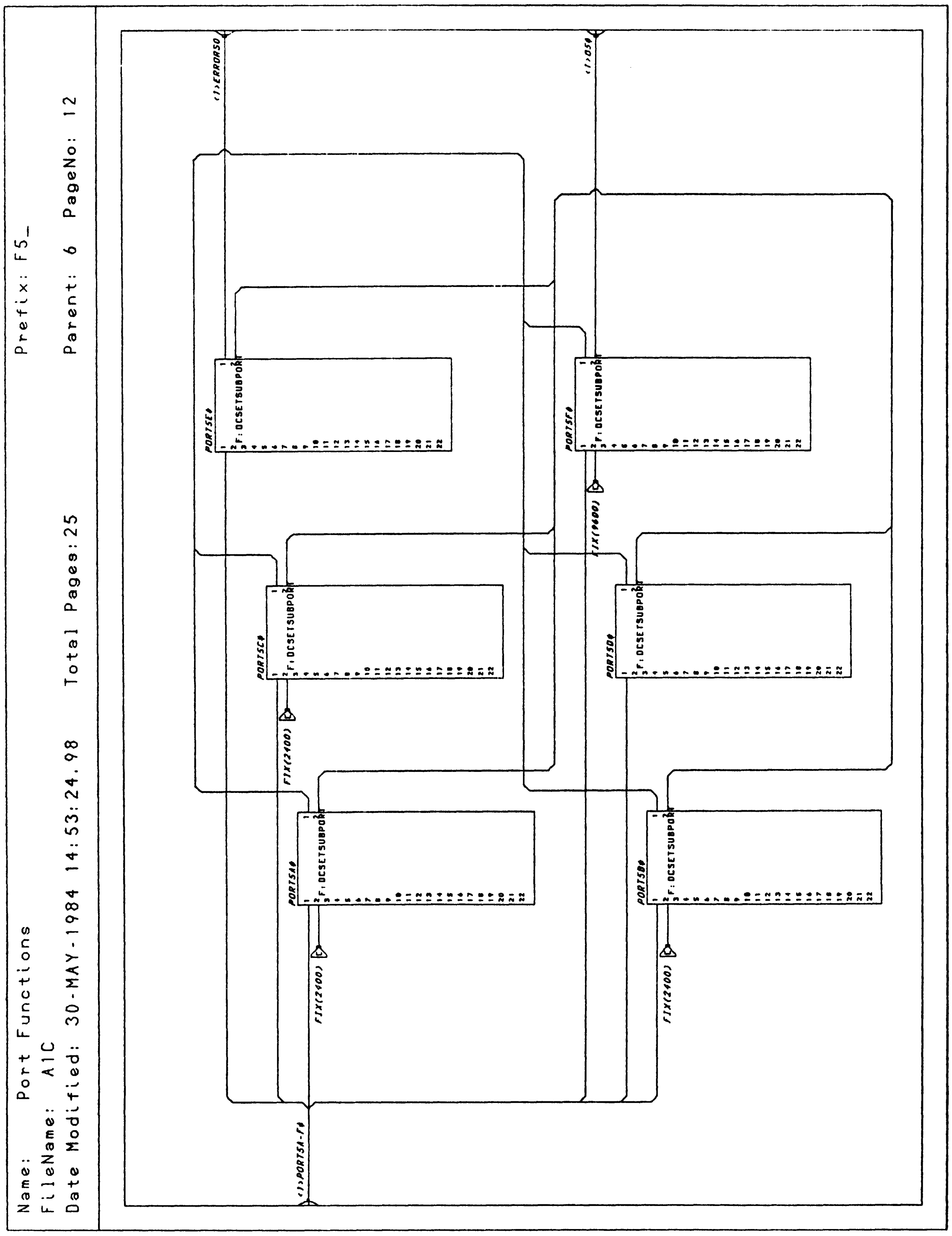


Figure 8-13. PS 390 Host Input Data Flow (RS-232 Interface)

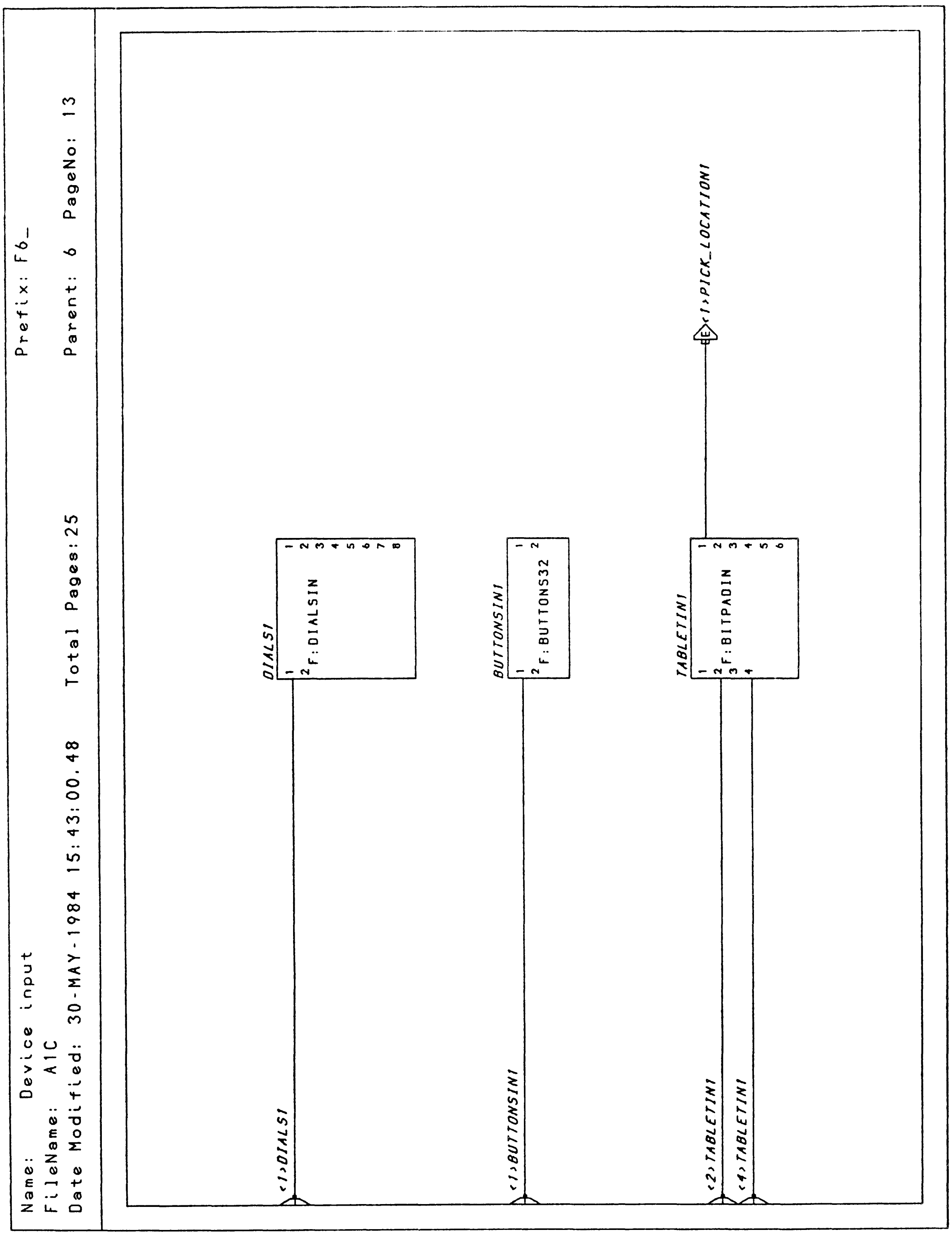


Figure 8-14. PS 390 Host Input Data Flow (RS-232 Interface)

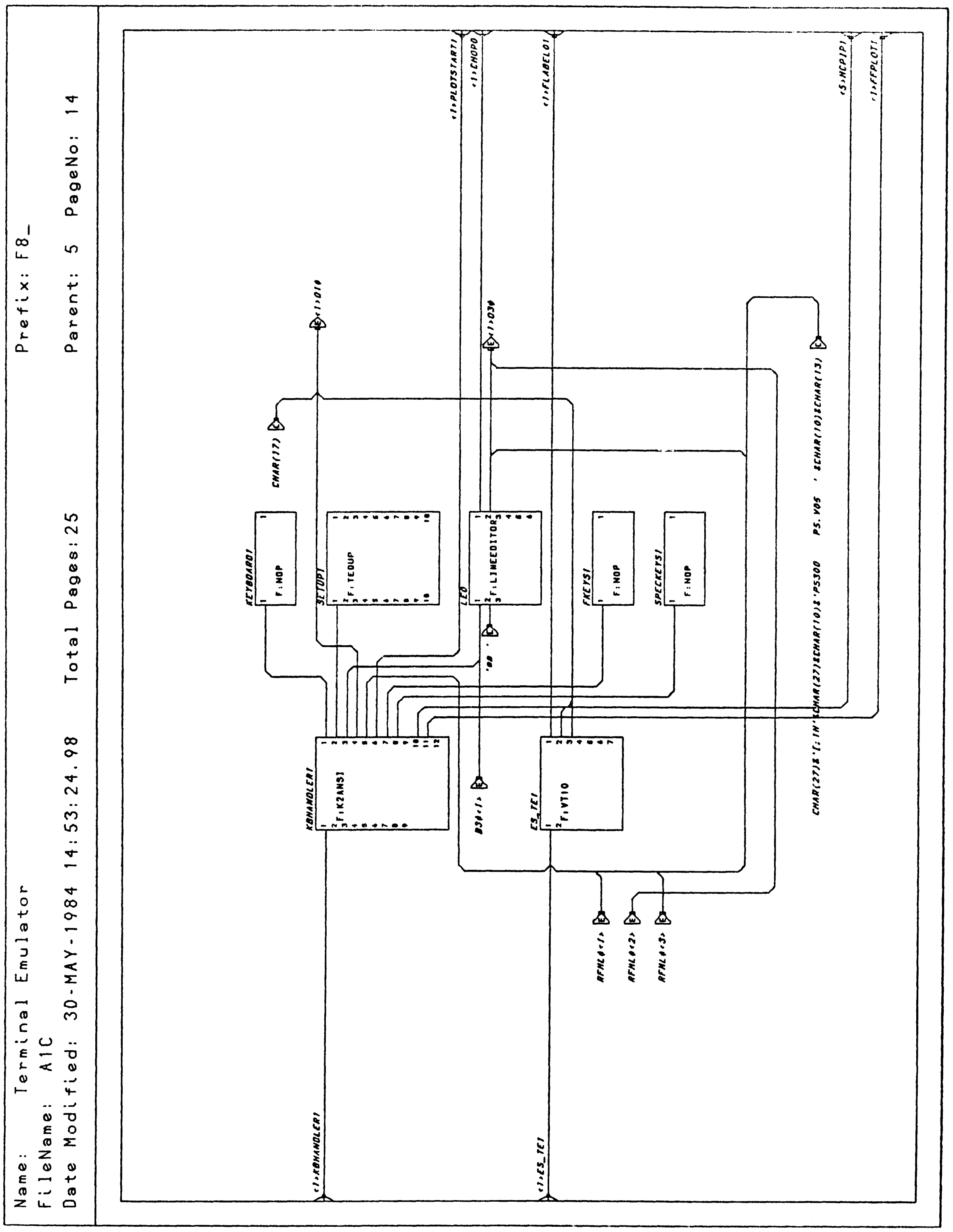


Figure 8-15. PS 390 Host Input Data Flow (RS-232 Interface)

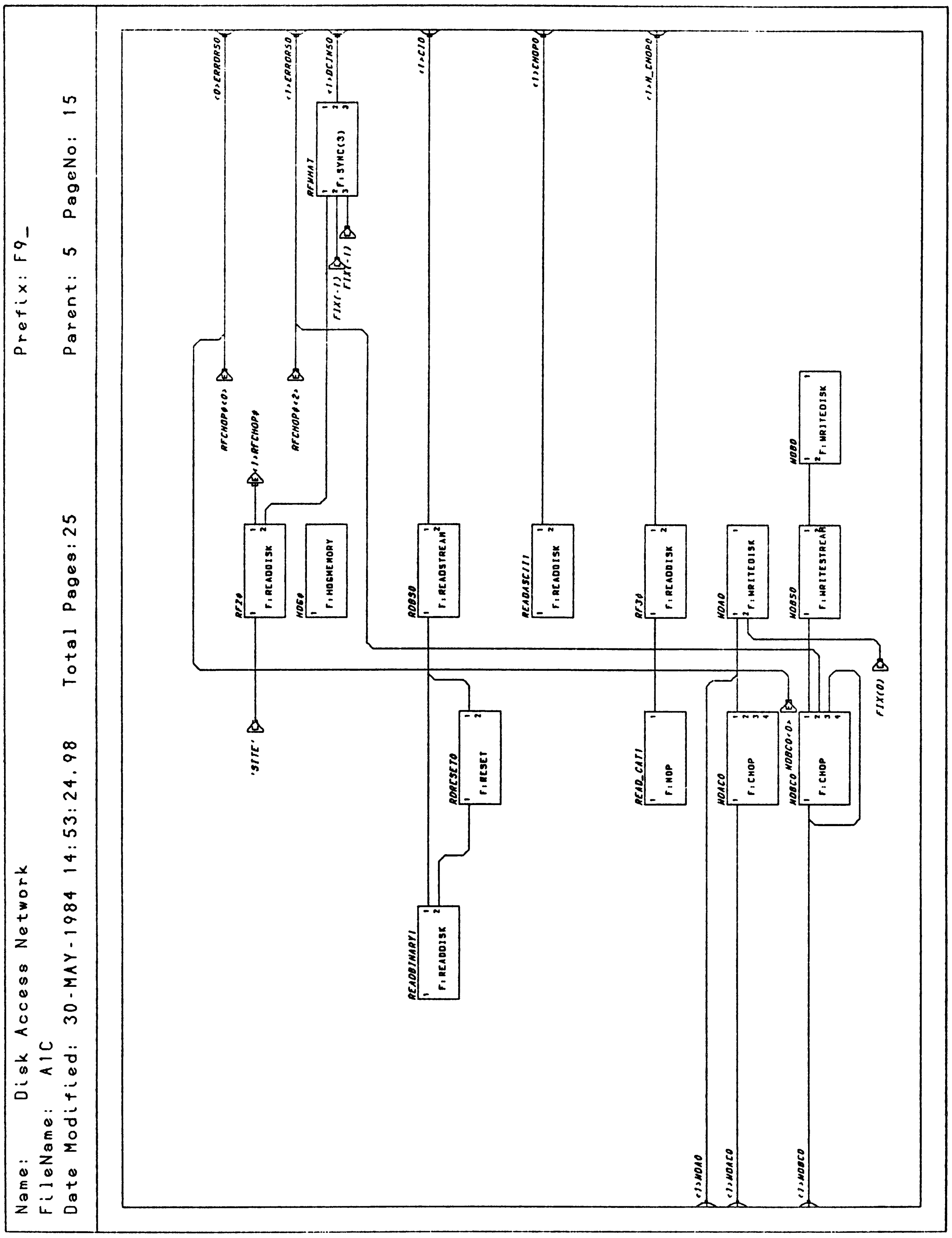


Figure 8-16. PS 390 Host Input Data Flow (RS-232 Interface)

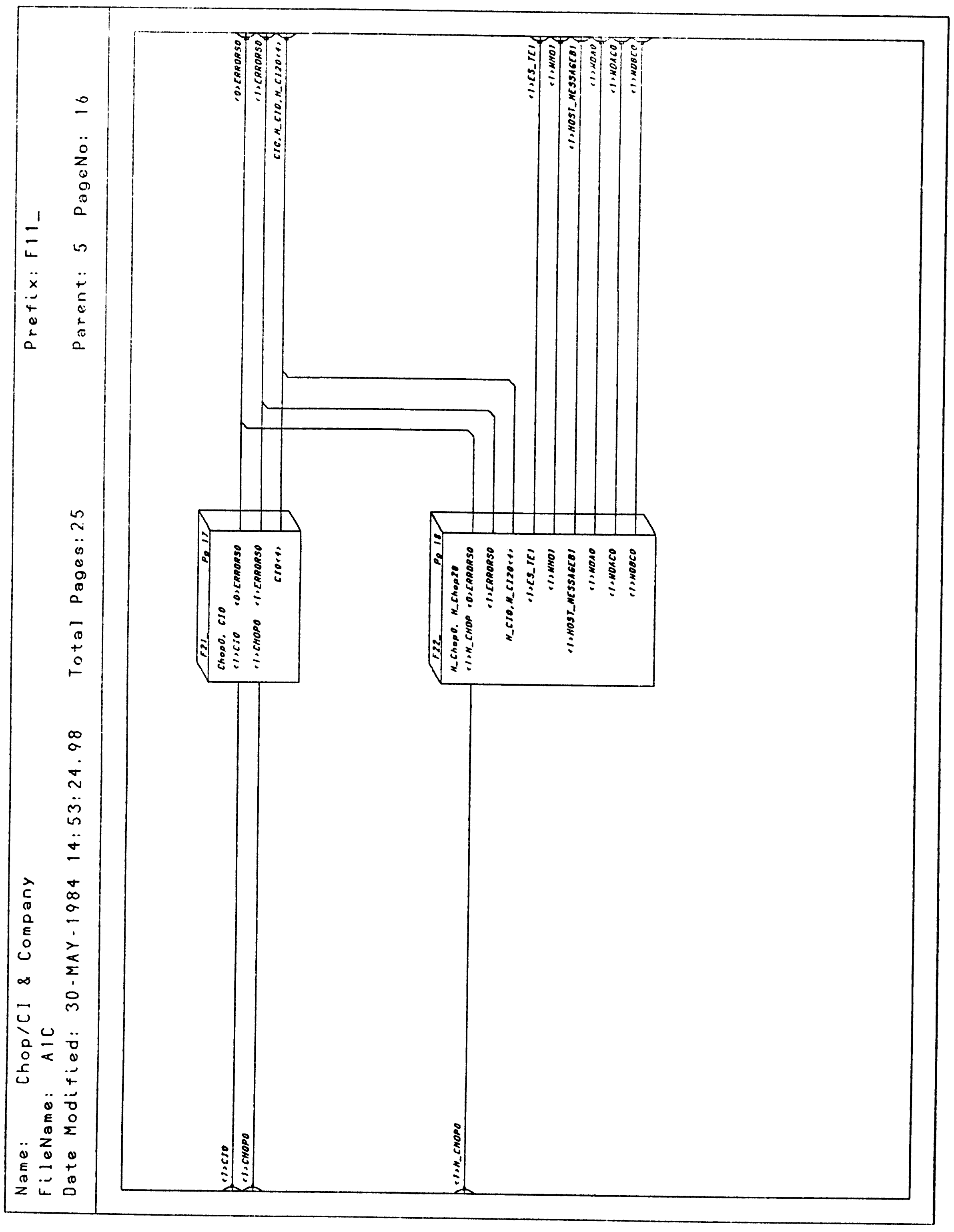


Figure 8-17. PS 390 Host Input Data Flow (RS-232 Interface)

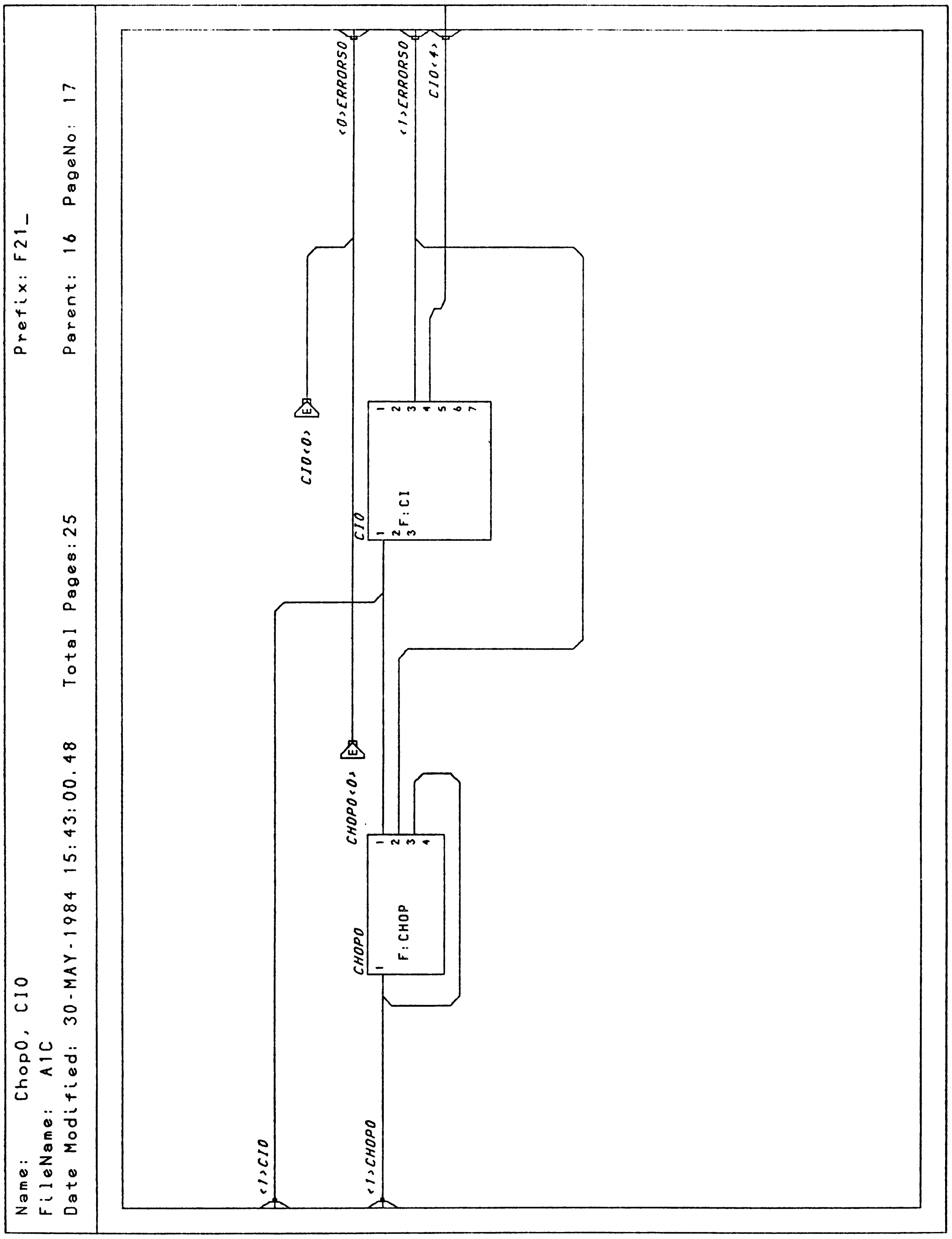


Figure 8-18. PS 390 Host Input Data Flow (RS-232 Interface)

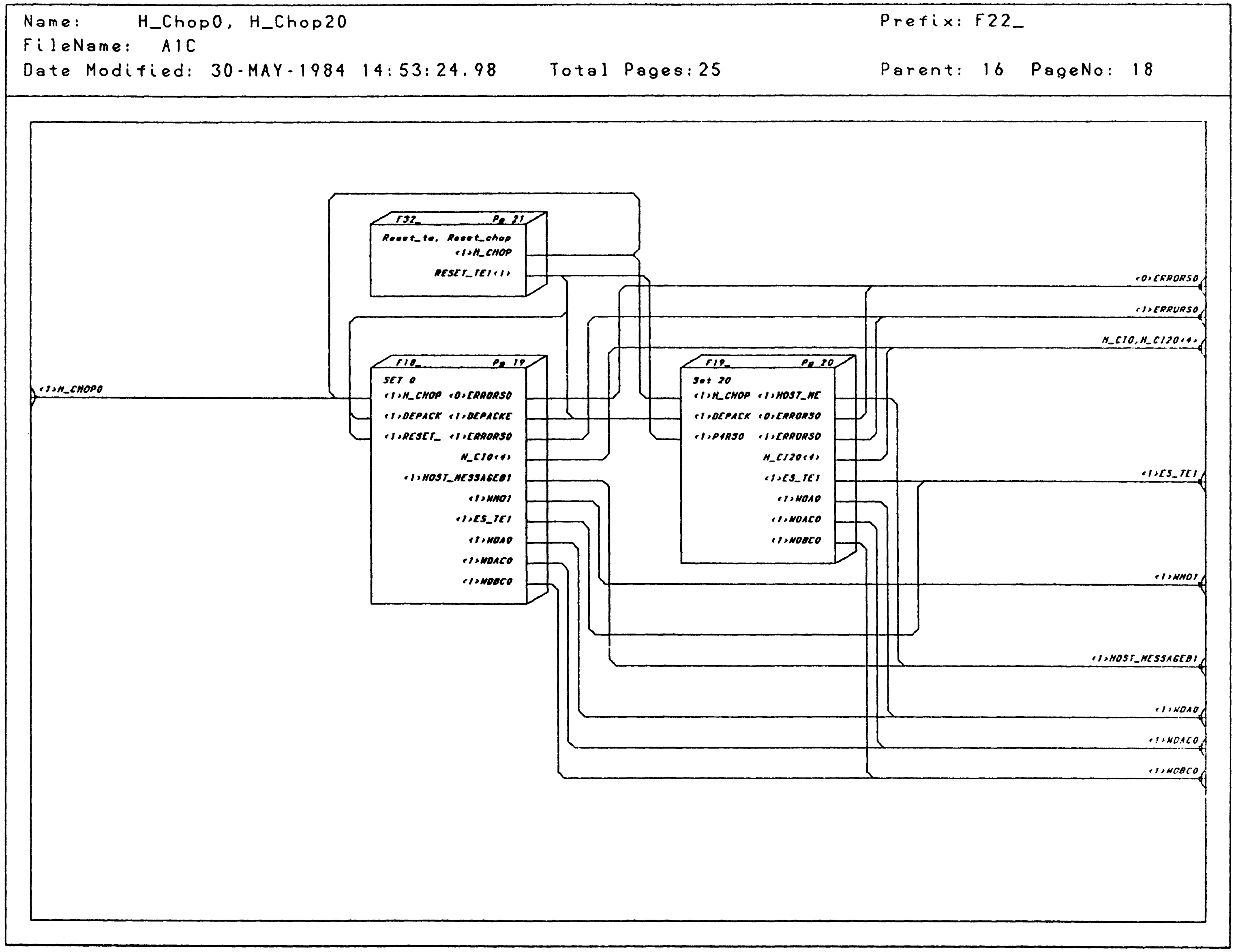


Figure 8-19. PS 390 Host Input Data Flow (RS-232 Interface)

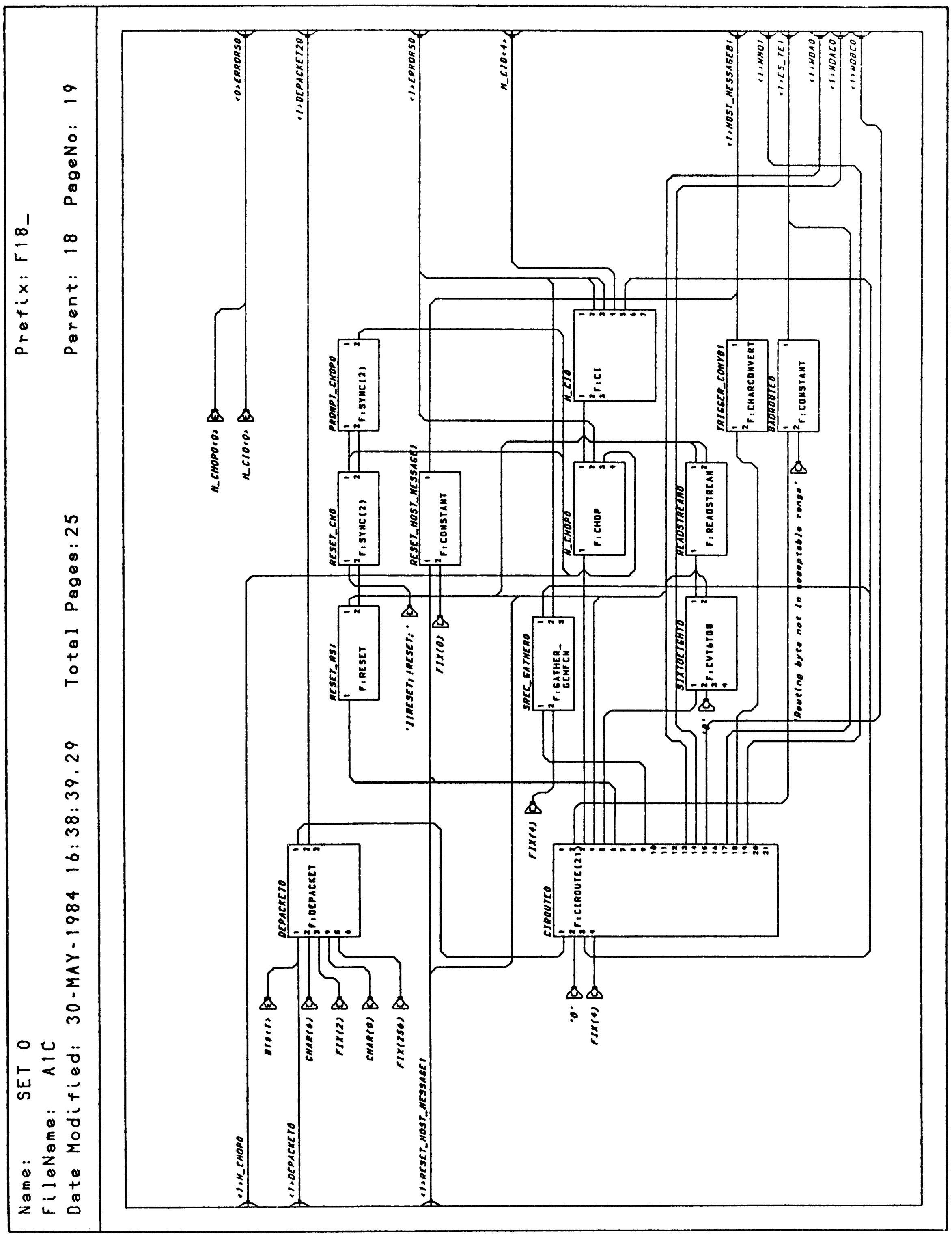


Figure 8-20. PS 390 Host Input Data Flow (RS-232 Interface)

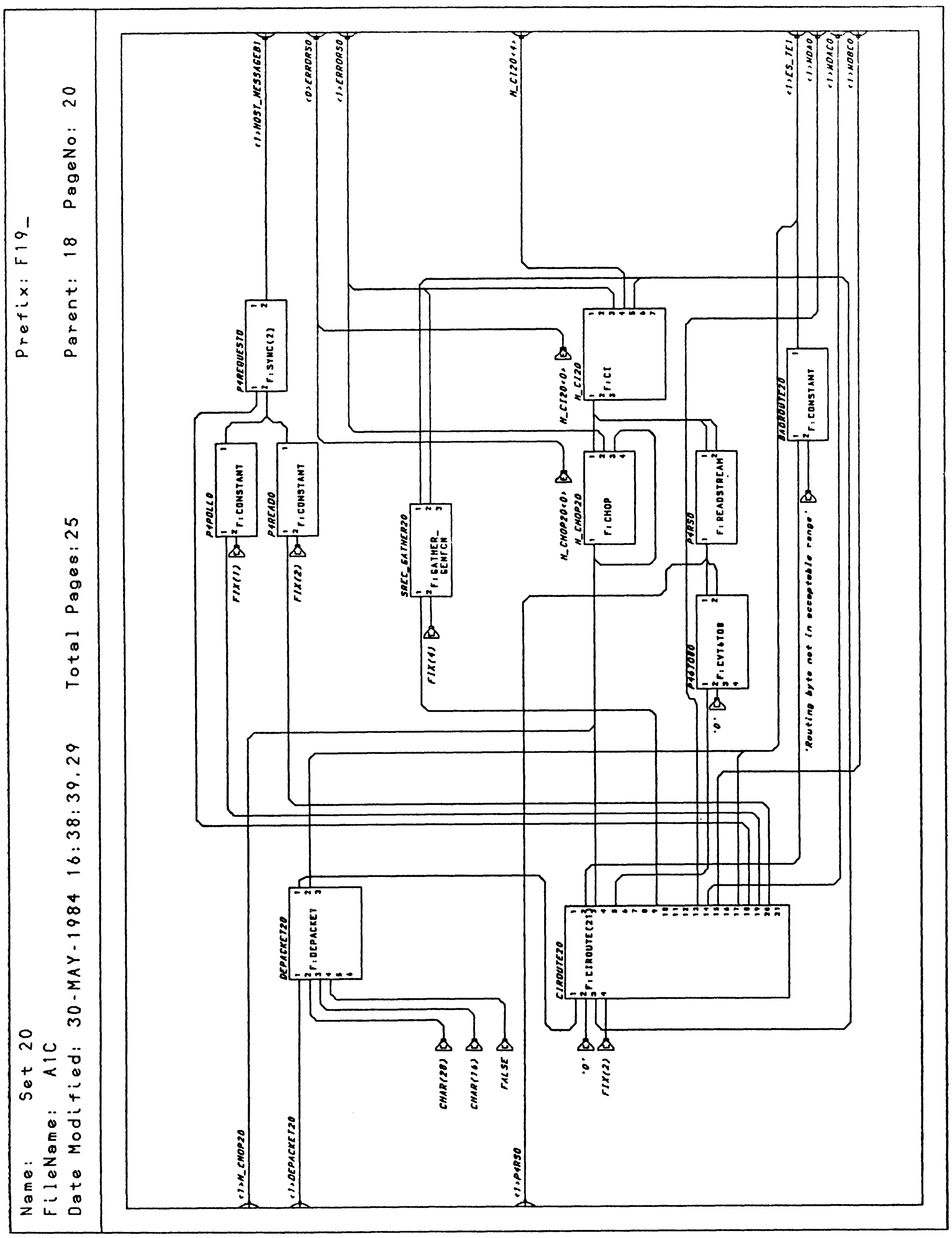


Figure 8-21. PS 390 Host Input Data Flow (RS-232 Interface)

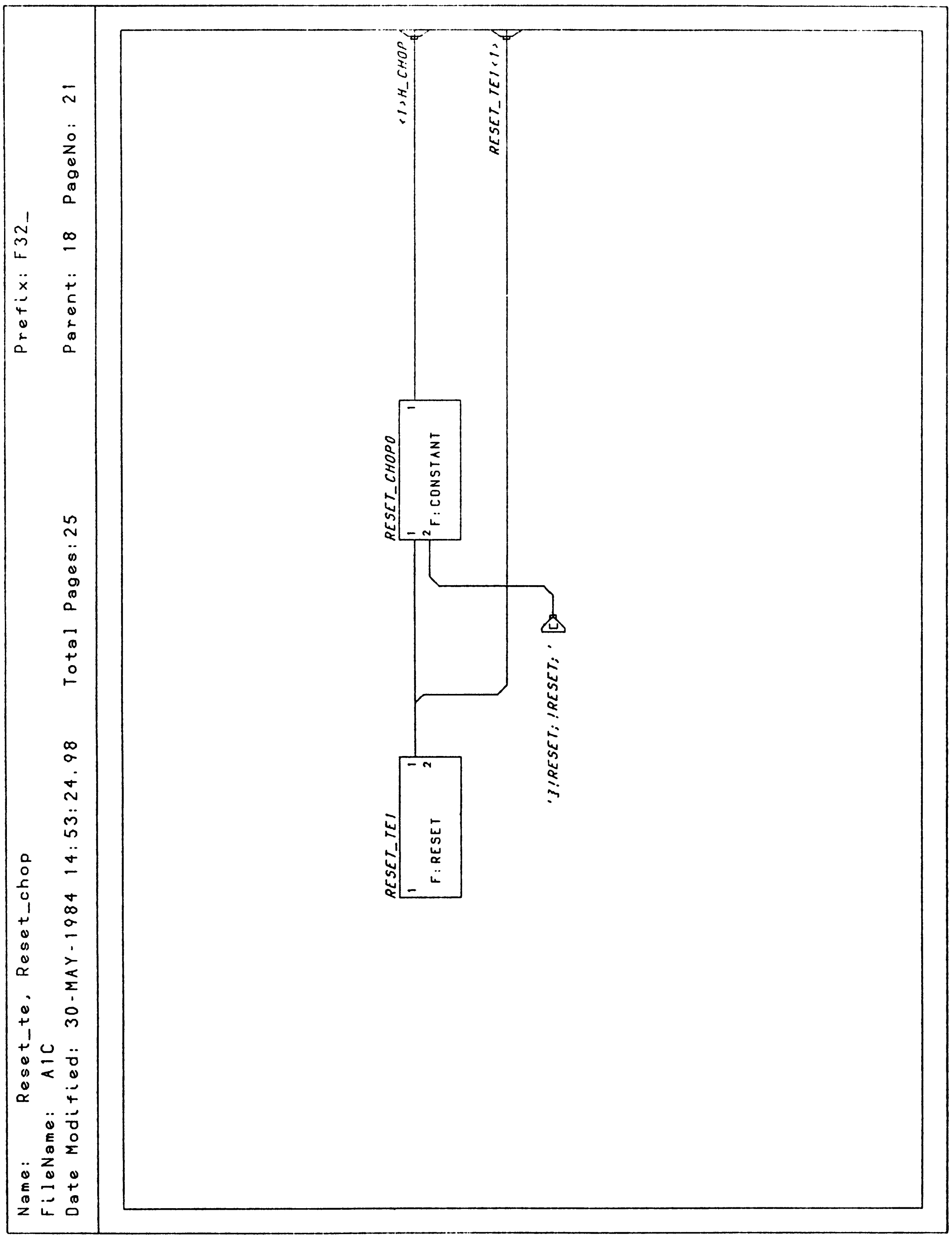


Figure 8-22. PS 390 Host Input Data Flow (RS-232 Interface)

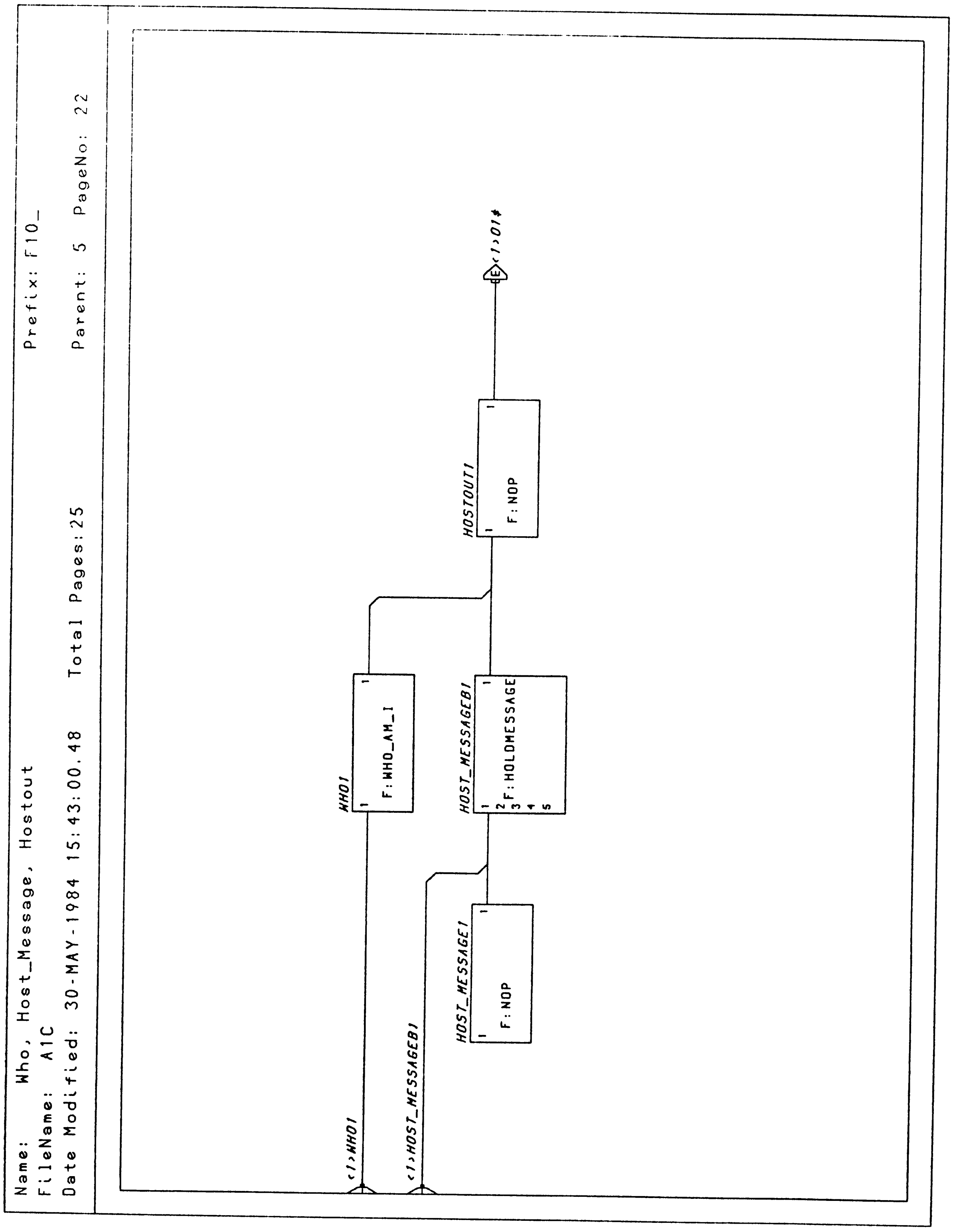


Figure 8-23. PS 390 Host Input Data Flow (RS-232 Interface)

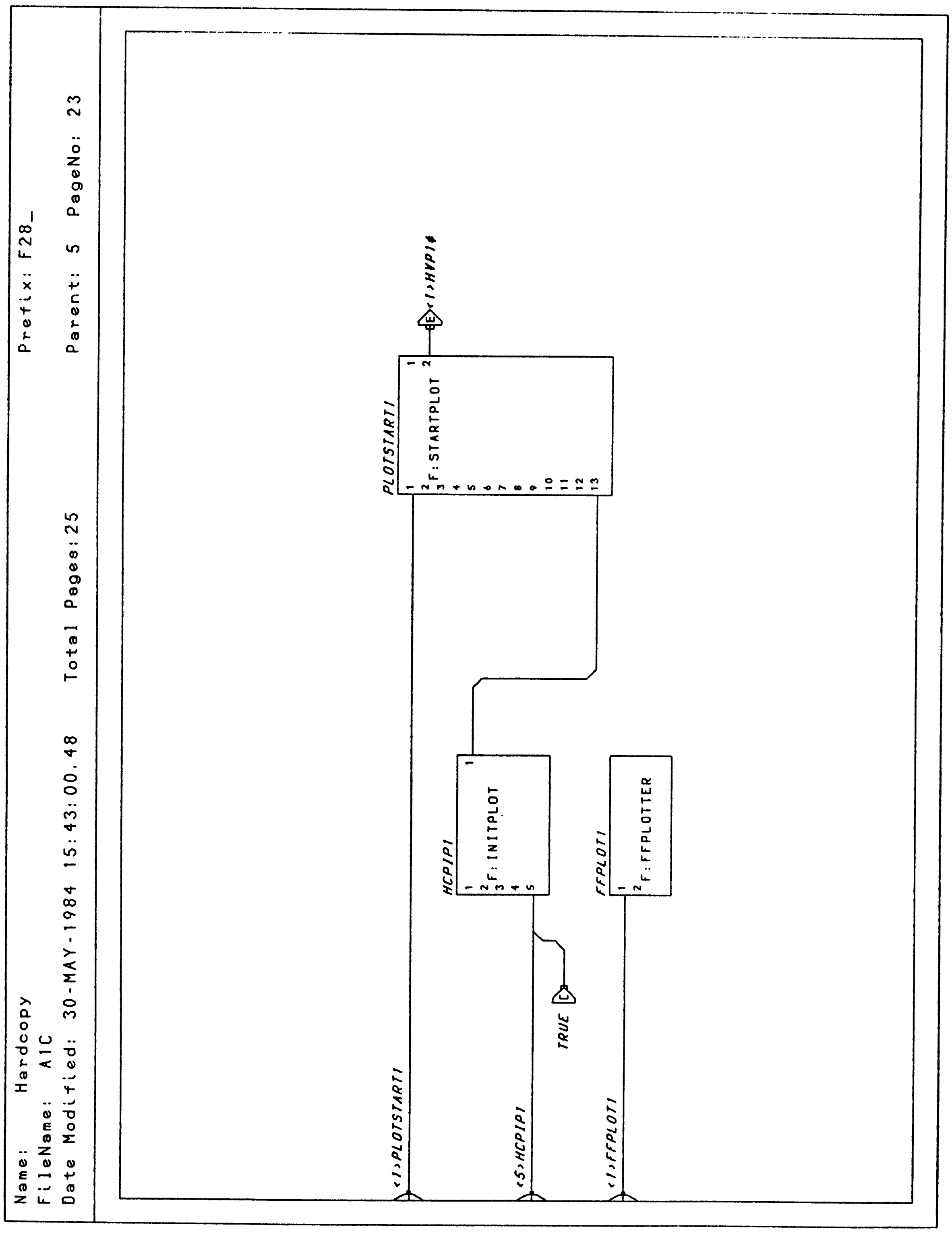


Figure 8-24. PS 390 Host Input Data Flow (RS-232 Interface)

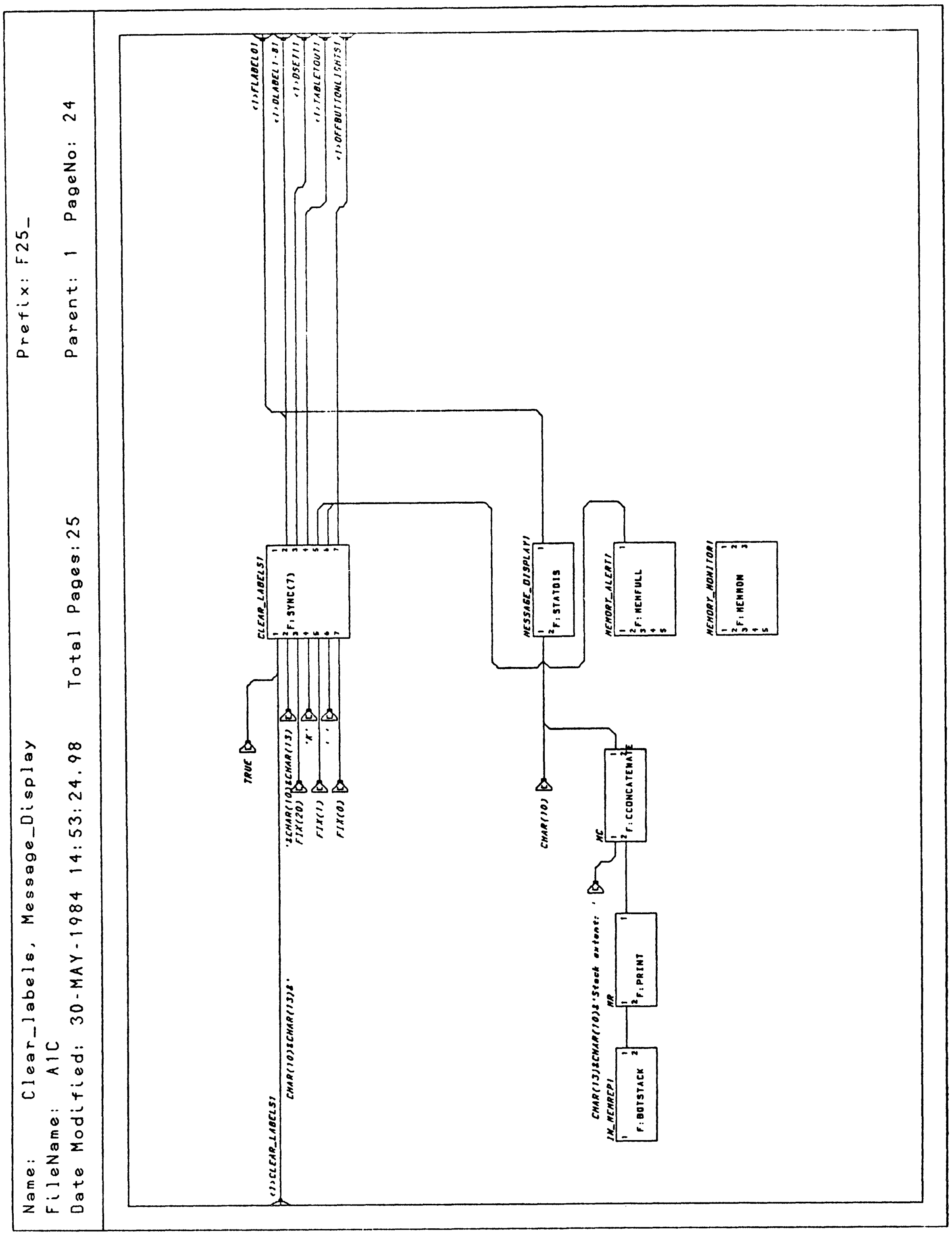


Figure 8-25. PS 390 Host Input Data Flow (RS-232 Interface)

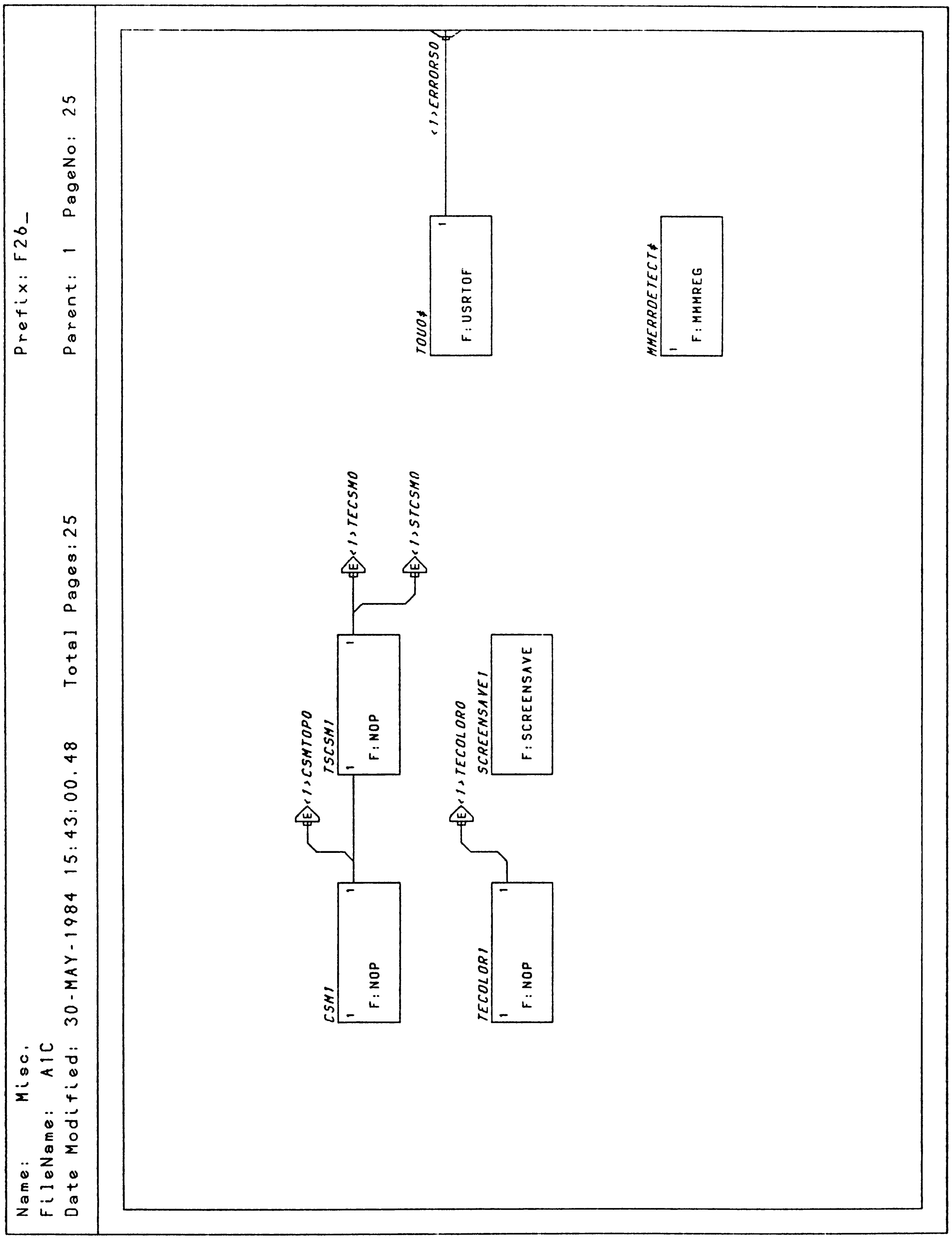


Figure 8-26. PS 390 Host Input Data Flow (RS-232 Interface)

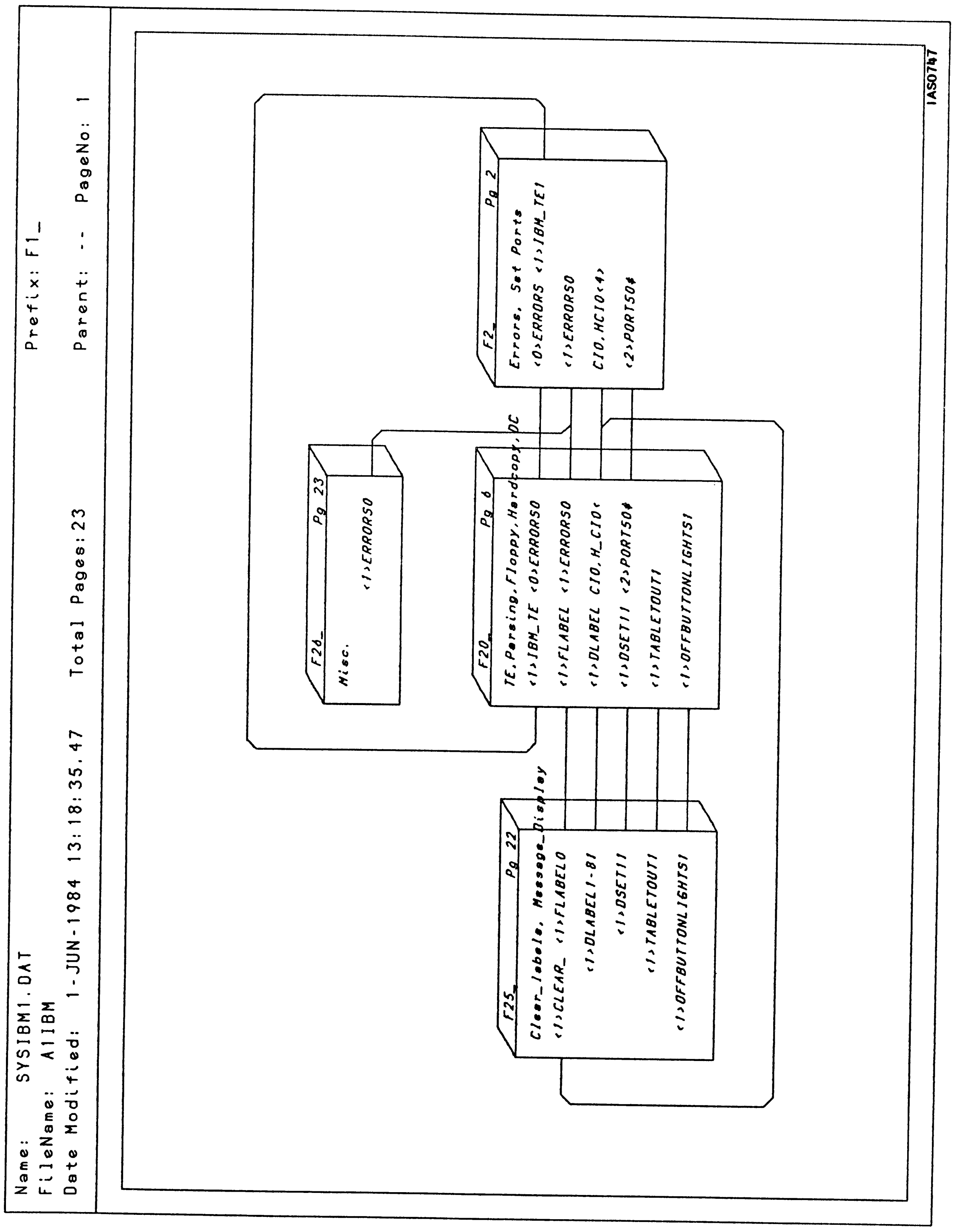


Figure 8-27. PS 390 Host Input Data Flow (IBM)

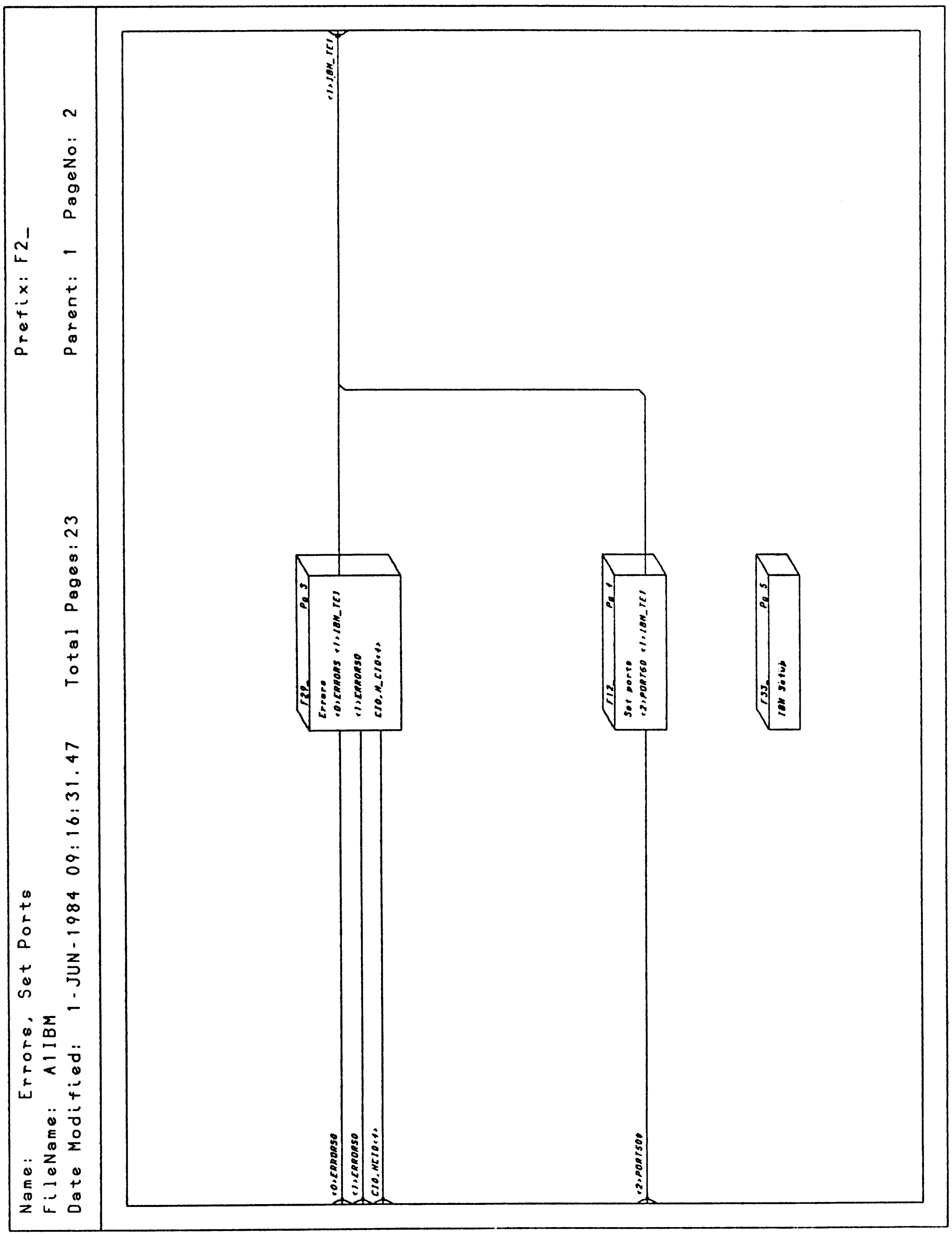


Figure 8-28. PS 390 Host Input Data Flow (IBM)

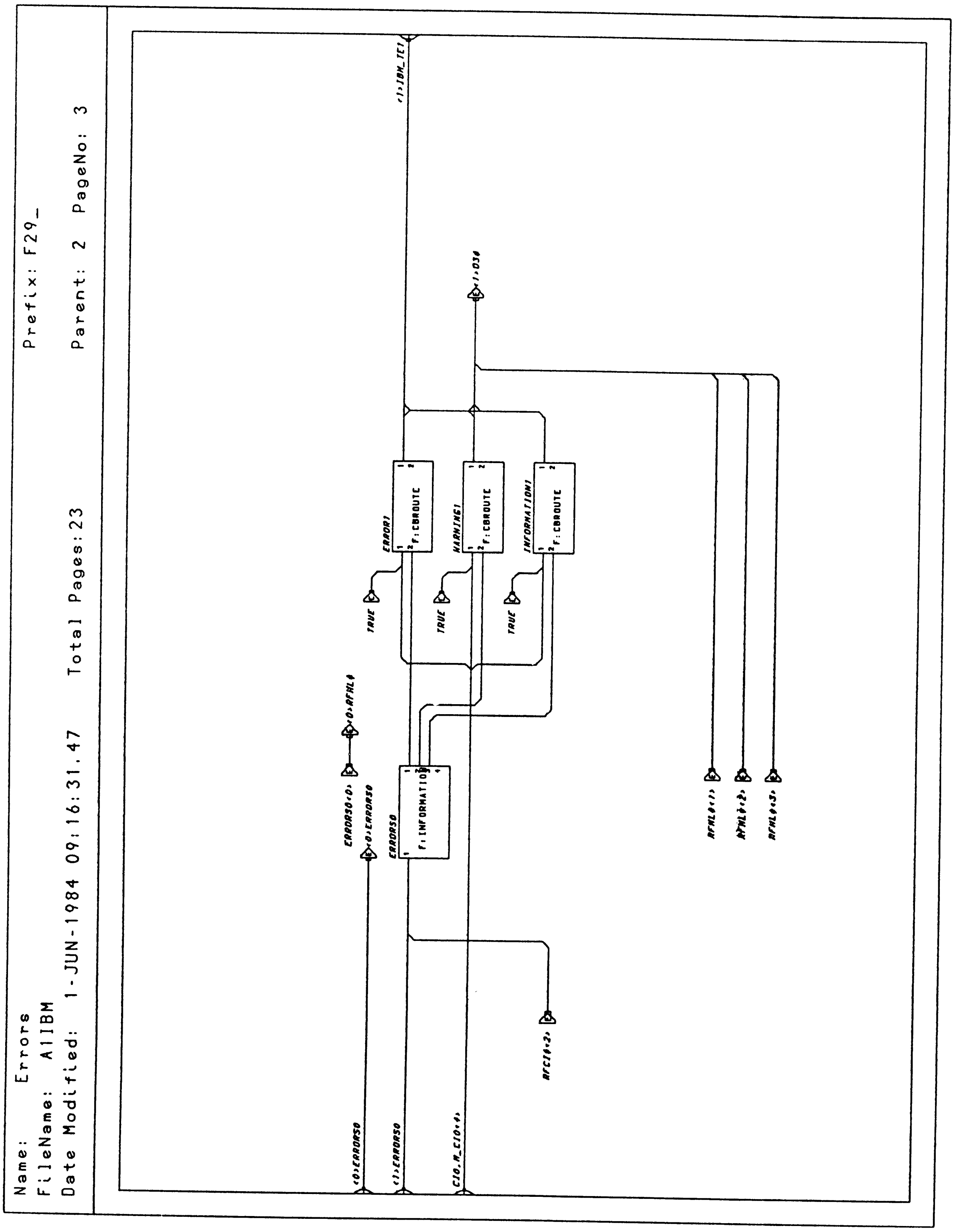


Figure 8-29. PS 390 Host Input Data Flow (IBM)

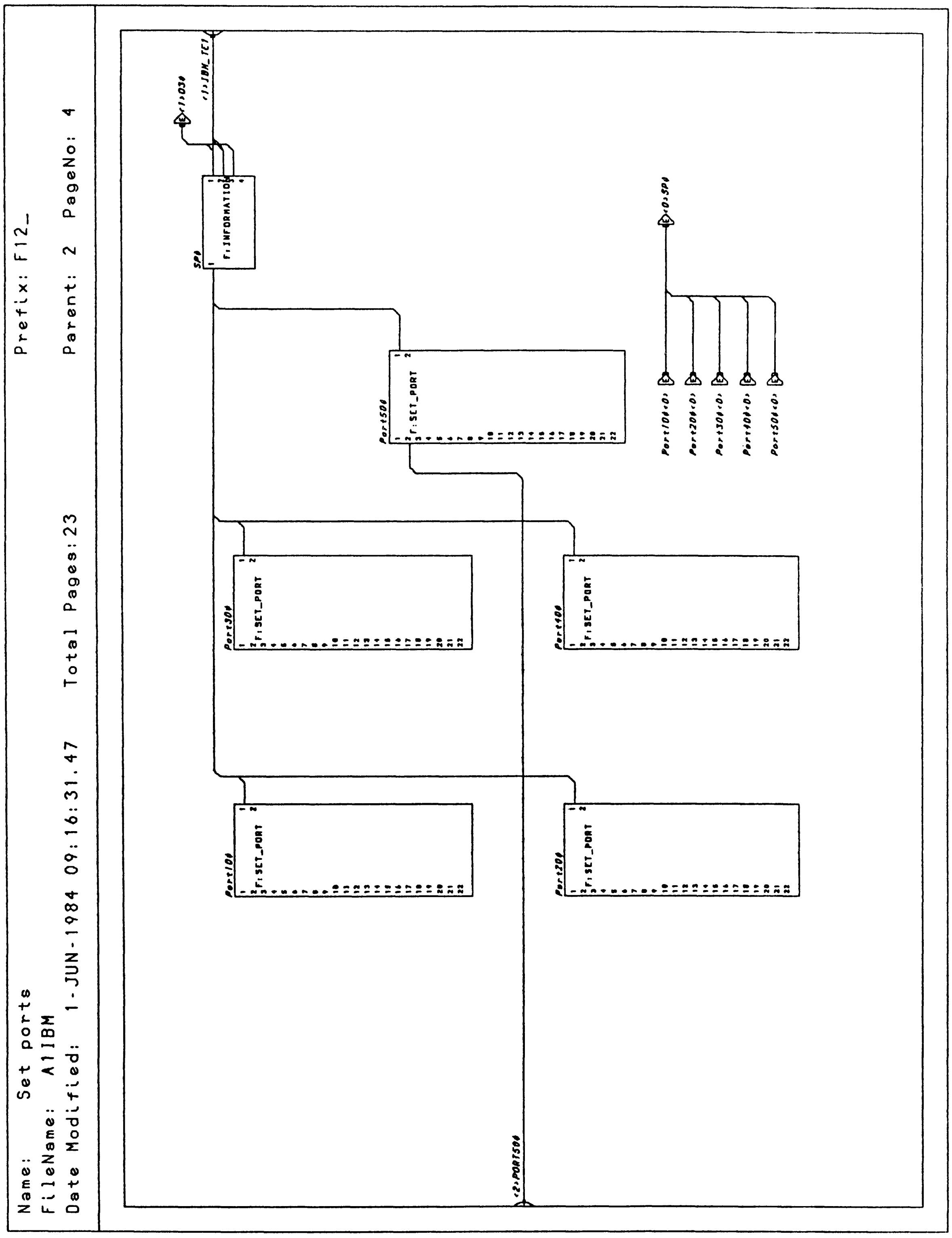


Figure 8-30. PS 390 Host Input Data Flow (IBM)

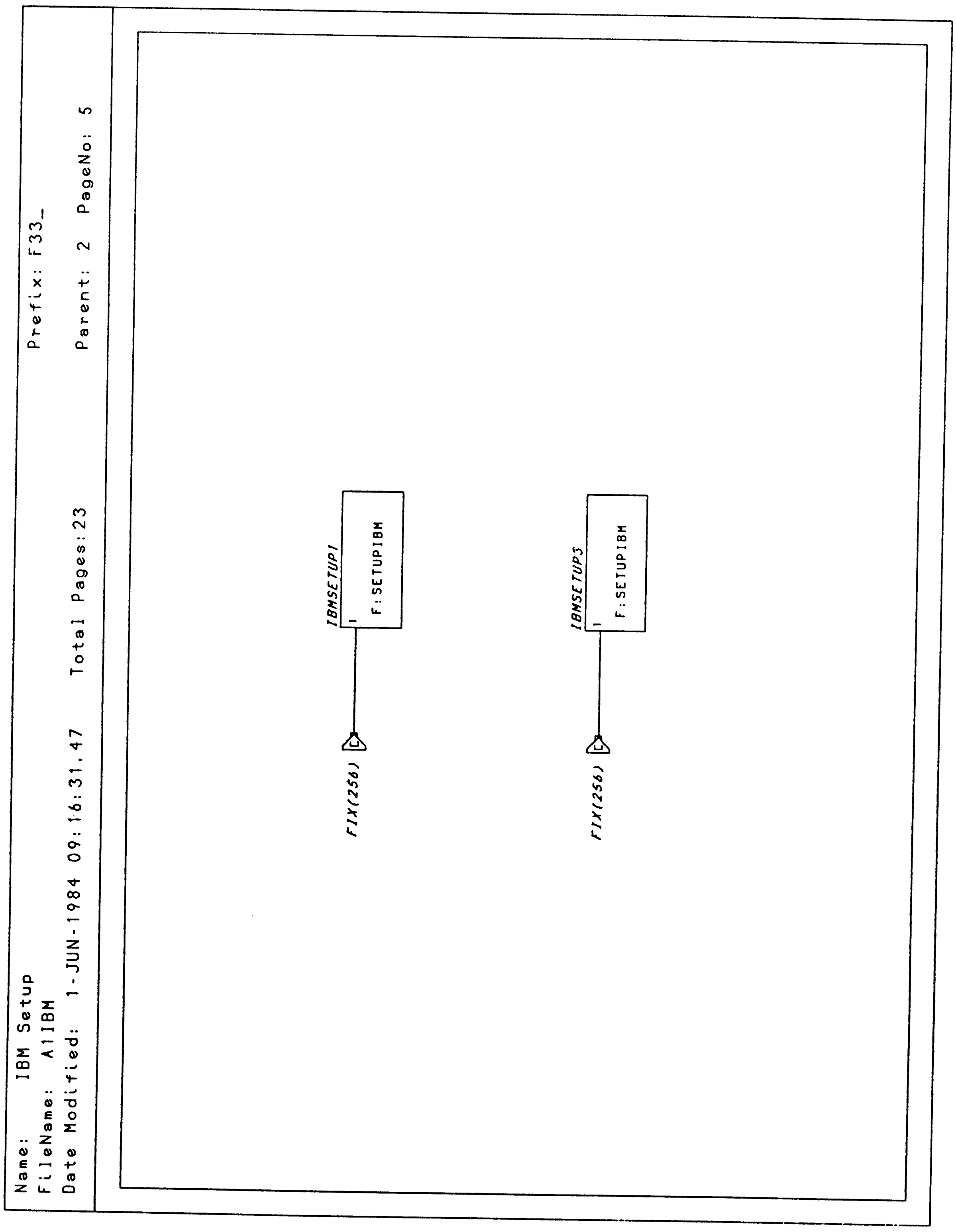
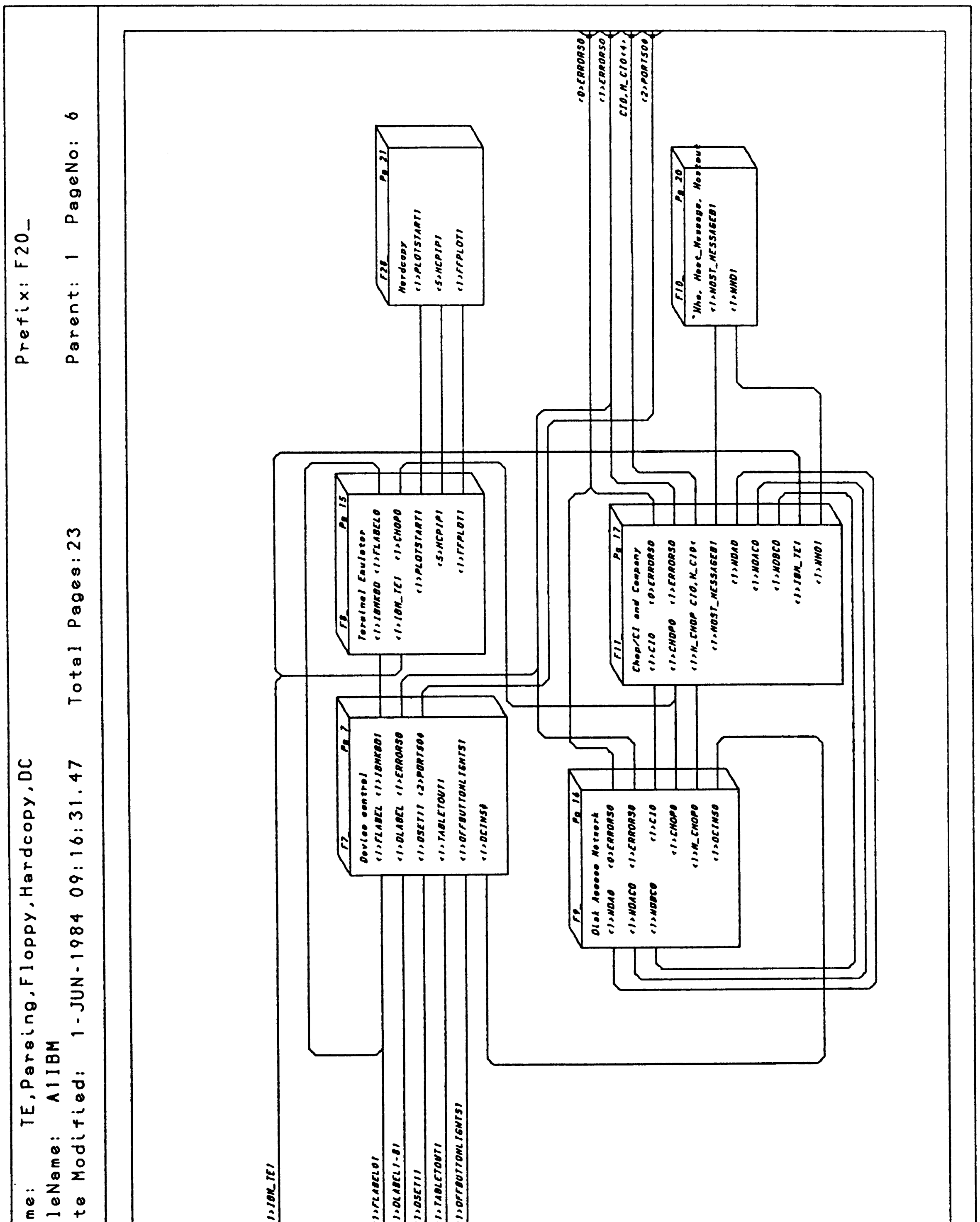


Figure 8-31. PS 390 Host Input Data Flow (IBM)



Name: TE,Parsing,Floppy,Hardcopy,DC
FileName: A11BM
Date Modified: 1-JUN-1984 09:16:31.47
Total Pages:23
Prefix: F20_
Parent: 1
PageNo: 6
F28 Pg 21 Hardcopy <1>PLOTSTART1 <5>HCPIP1 <1>FFPLOT1
F10 Pg 20 "Who, Host_Message, Hostout <1>HOST_MESSAGEB1 <1>WHO1
F8 Pg 15 Terminal Emulator <1>IBMKBD <1>FLABEL0 <1>IBM_TE1 <1>CHOP0 <1>PLOTSTART1 <5>HCPIP1 <1>FFPLOT1
F11 Pg 17 Chop/CI and Company <1>CI0 <0>ERRORS0 <1>CHOP0 <1>ERRORS0 <1>H_CHOP CI0,H_CI0 <1>HOST_MESSAGEB1 <1>HDA0 <1>HDAC0 <1>HDBC0 <1>IBM_TE1 <1>WHO1
F7 Pg 7 Device control <1>FLABEL <1>IBMKBD1 <1>DLABEL <1>ERRORS0 <1>DSET11 <2>PORTS00 <1>TABLETOUT1 <1>OFFBUTTONLIGHTS1 <1>DCINS0
F9 Pg 16 Disk Access Network <1>HDA0 <0>ERRORS0 <1>HDAC0 <1>ERRORS0 <1>HDBC0 <1>CI0 <1>CHOP0 <1>H_CHOP0 <1>DCINS0
<0>ERRORS0
<1>ERRORS0
CI0,H_CI0<4>
<2>PORTS00
<1>IBM_TE1
<1>FLABEL01
<1>DLABEL1-81
<1>DSET11
<1>TABLETOUT1
<1>OFFBUTTONLIGHTS1

Figure 8-32. PS 390 Host Input Data Flow (IBM)

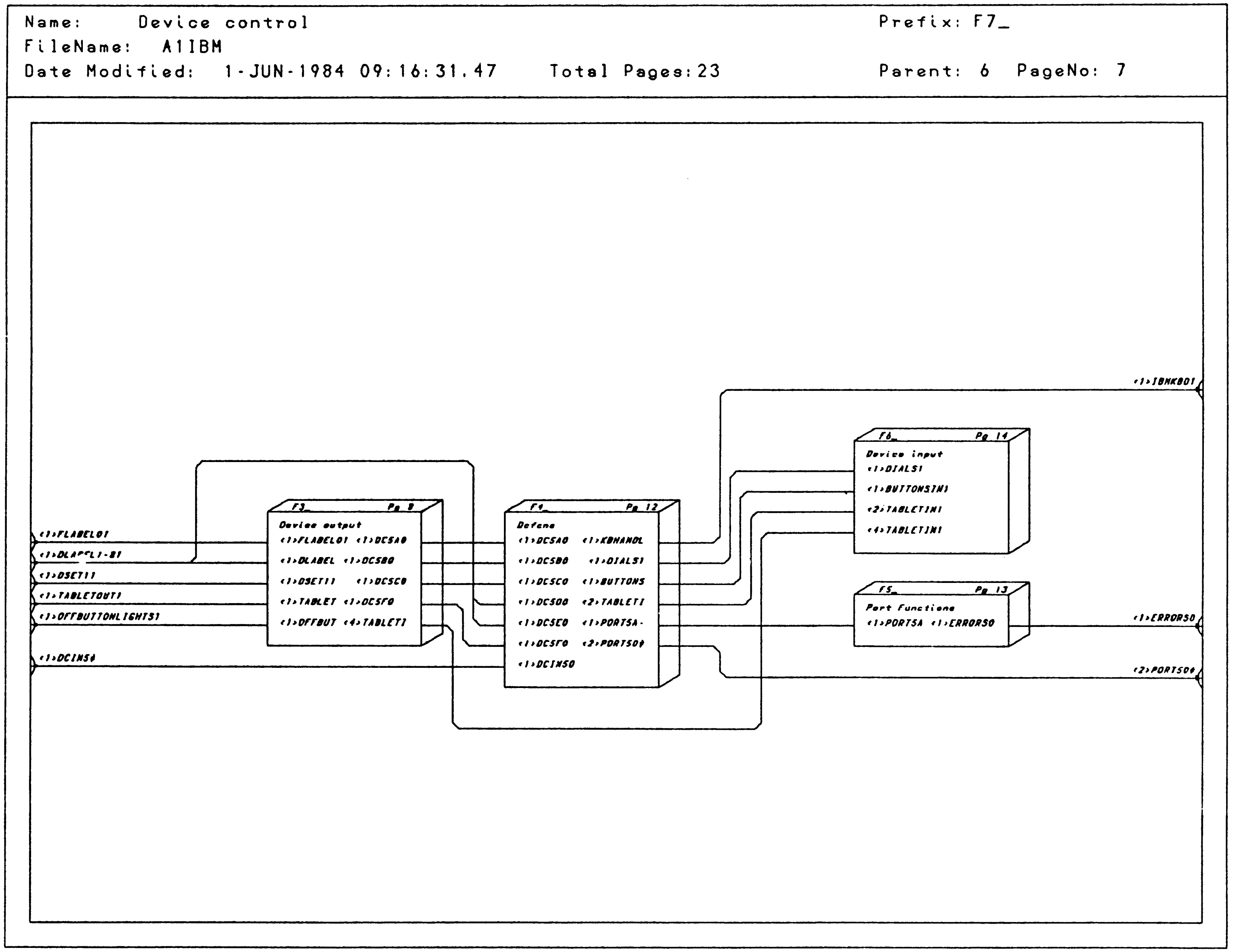


Figure 8-33. PS 390 Host Input Data Flow (IBM)

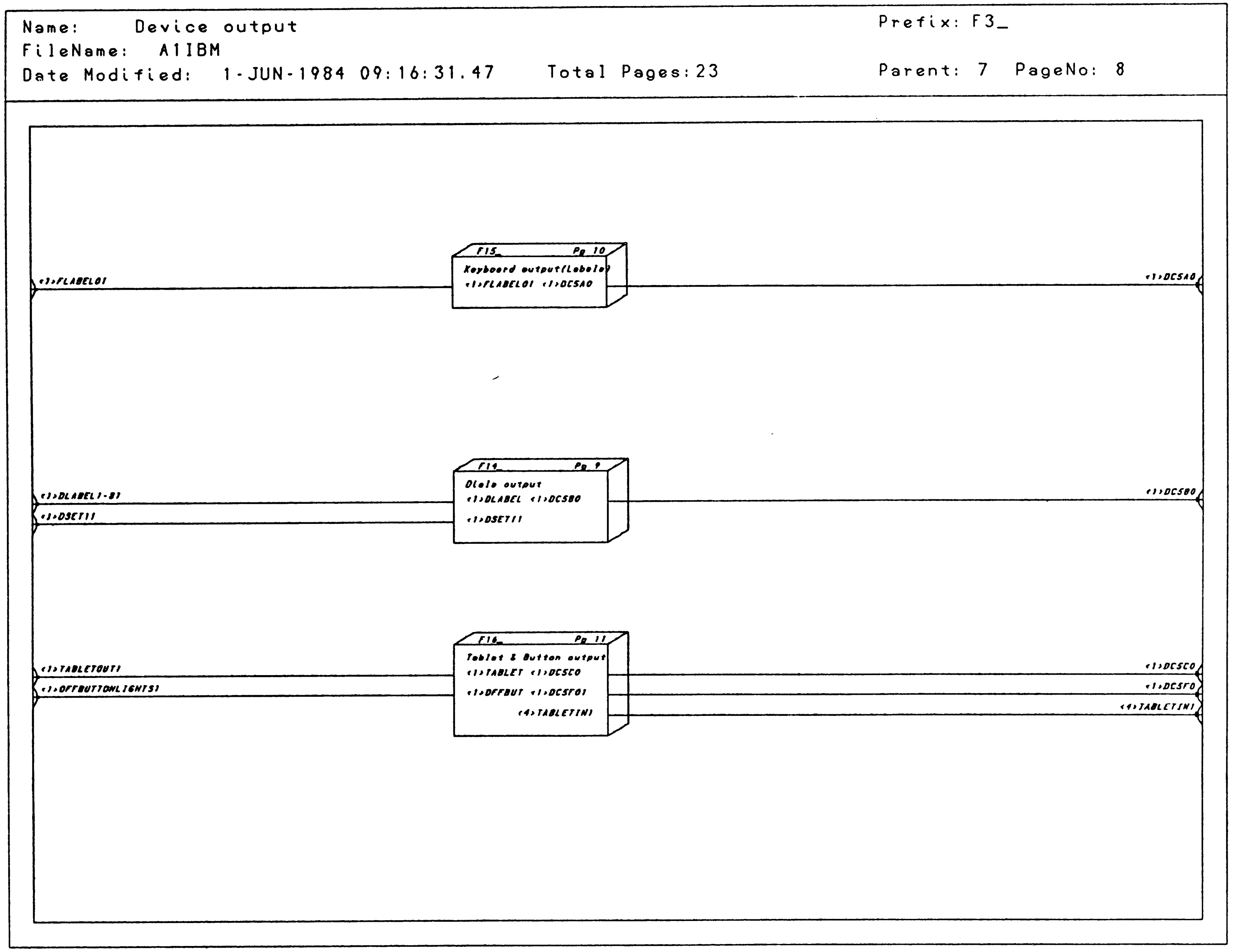


Figure 8-34. PS 390 Host Input Data Flow (IBM)

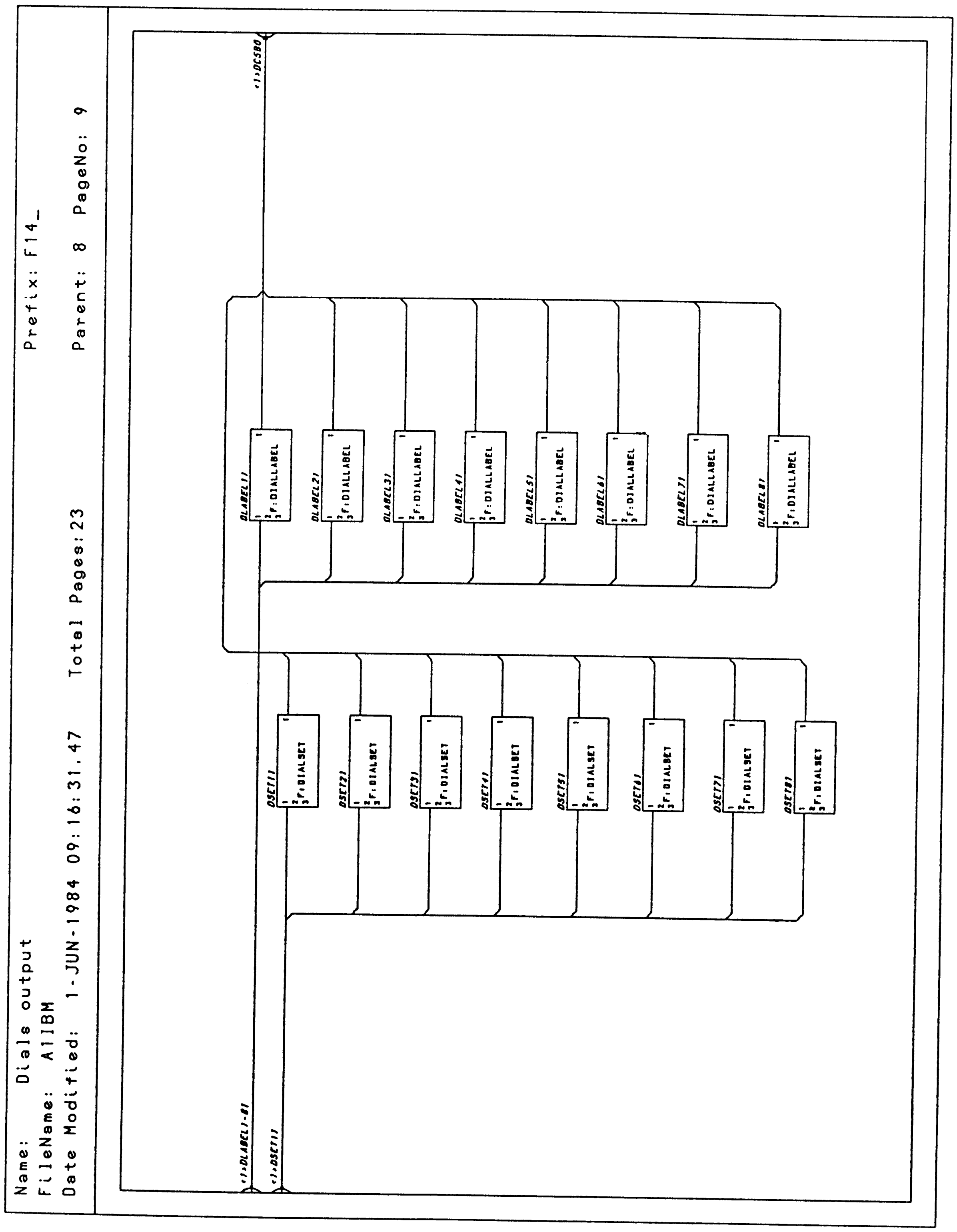


Figure 8-35. PS 390 Host Input Data Flow (IBM)

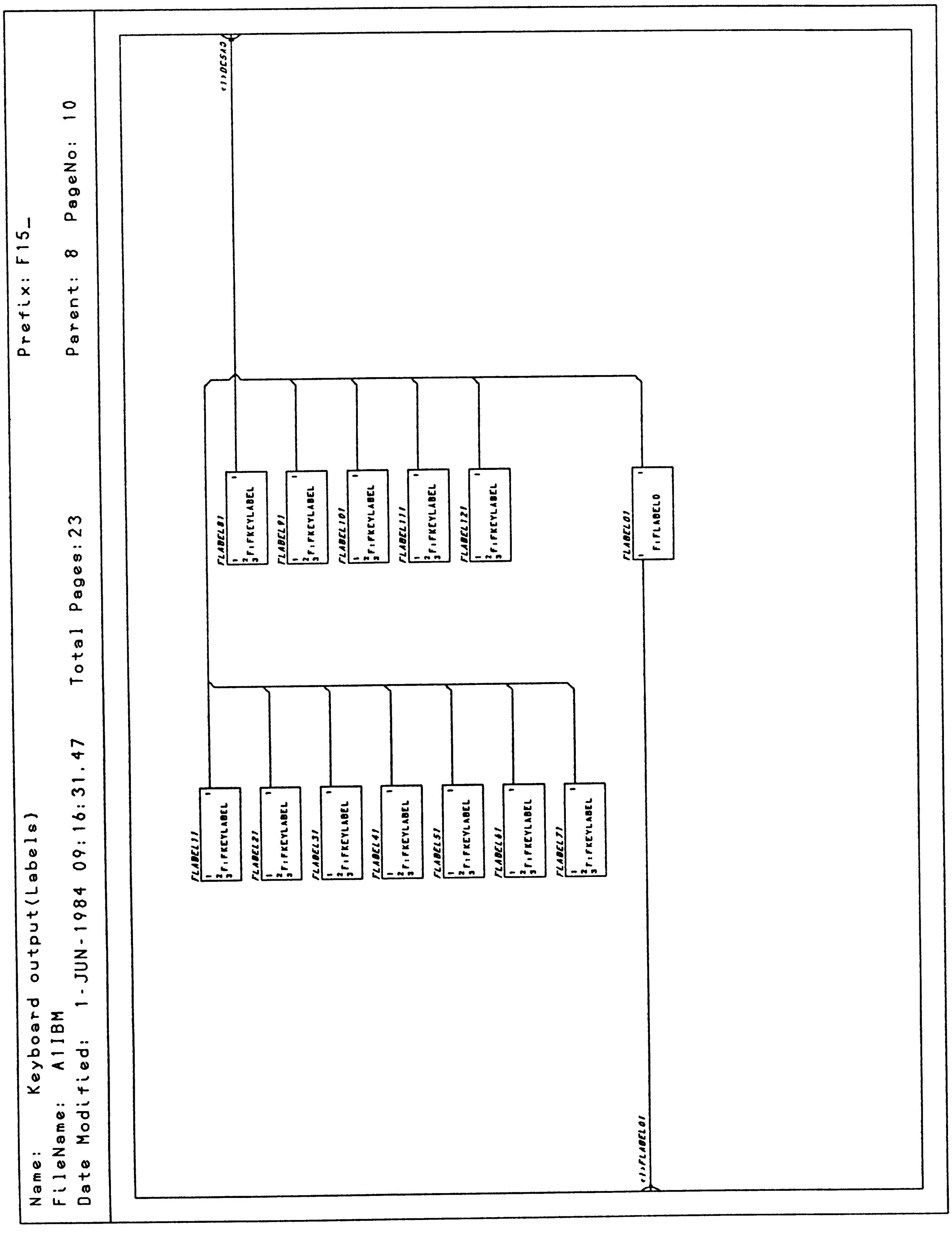


Figure 8-36. PS 390 Host Input Data Flow (IBM)

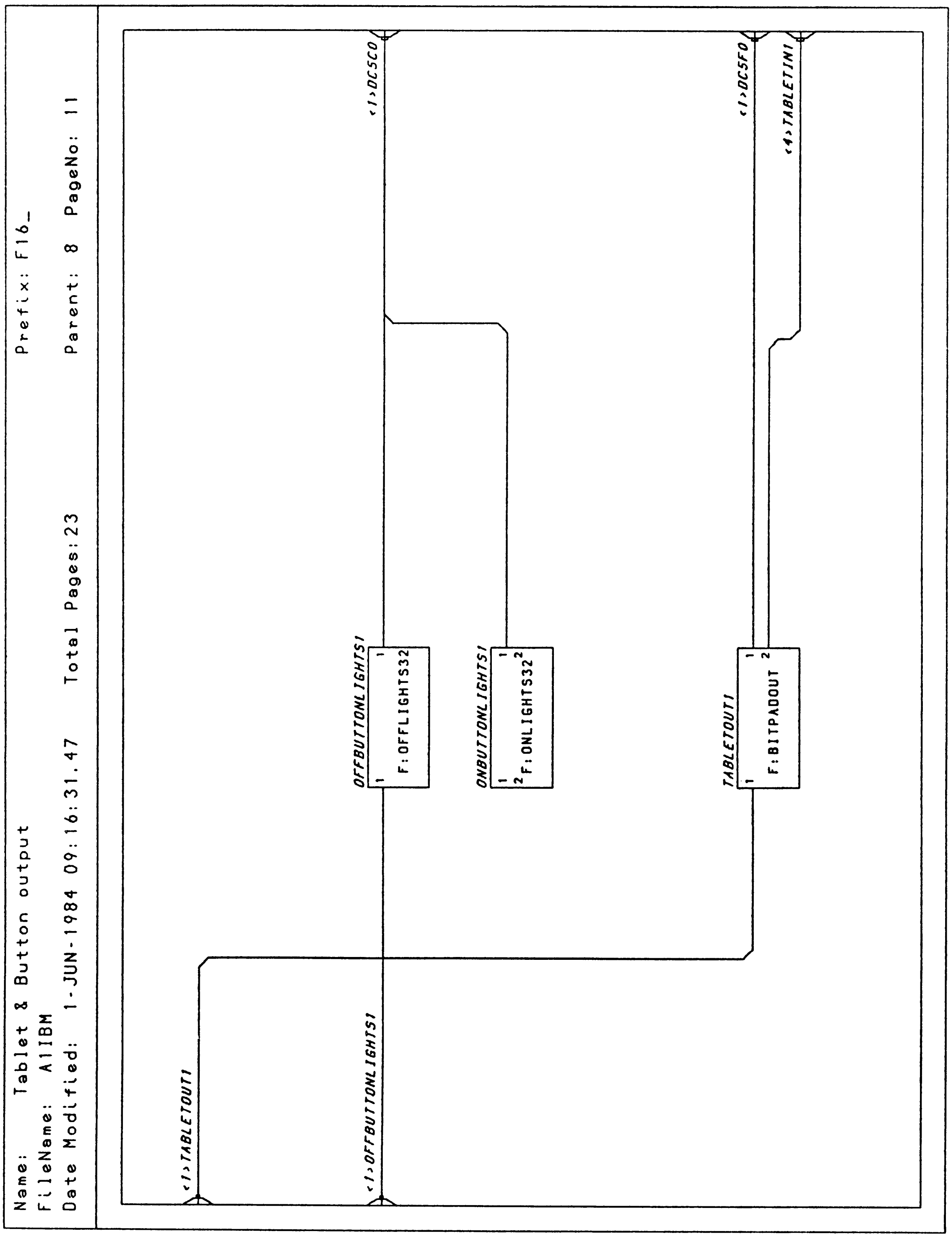


Figure 8-37. PS 390 Host Input Data Flow (IBM)

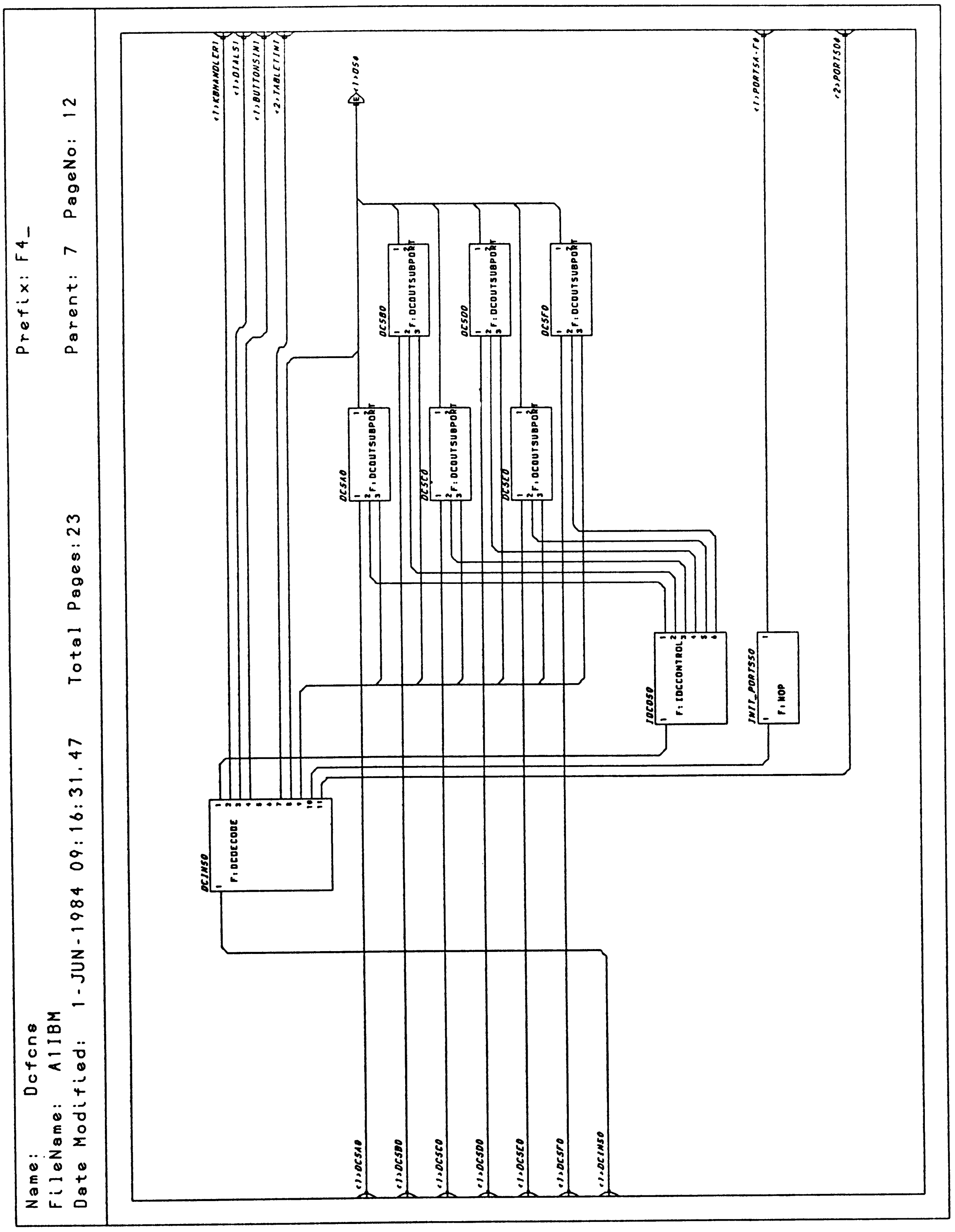


Figure 8-38. PS 390 Host Input Data Flow (IBM)

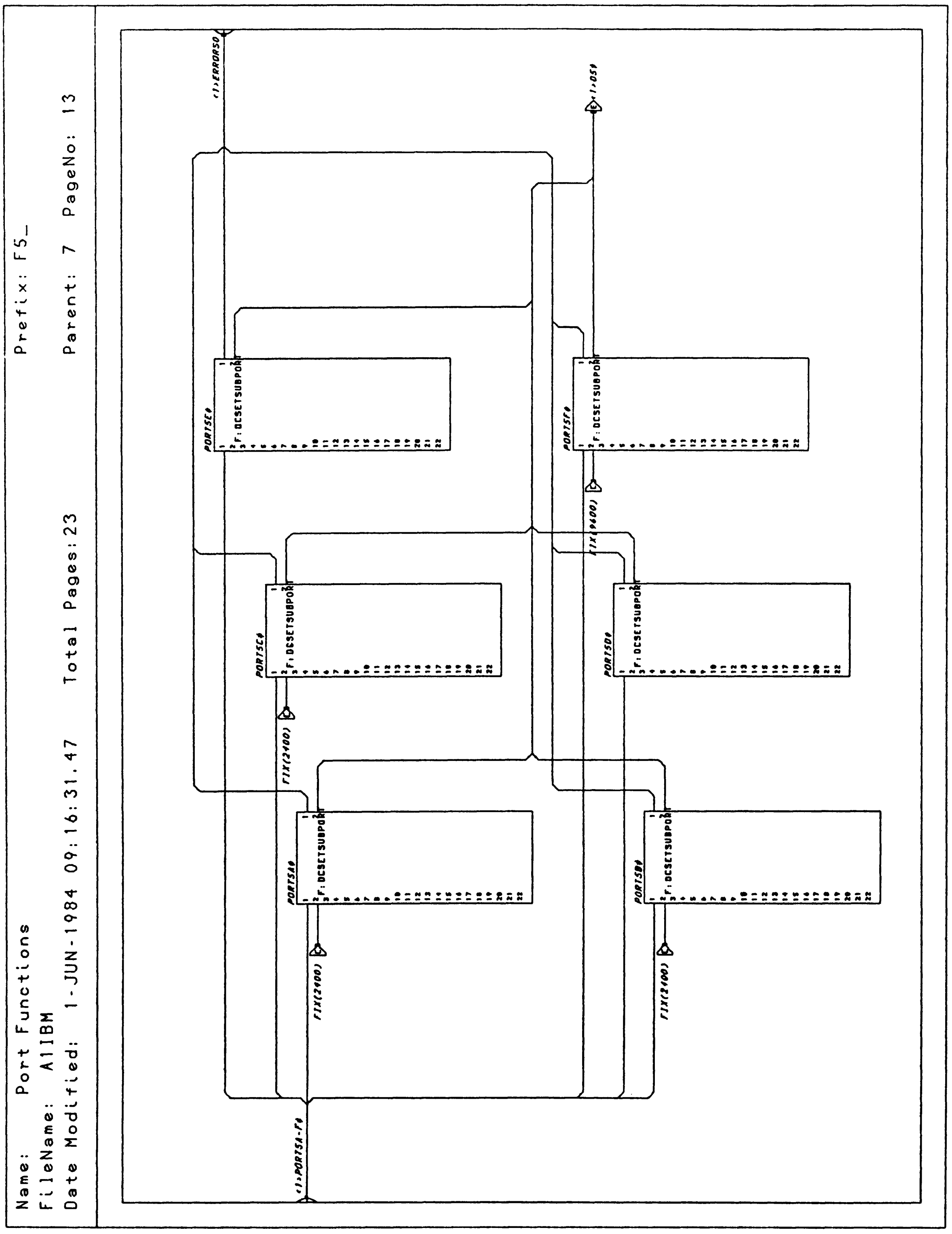


Figure 8-39. PS 390 Host Input Data Flow (IBM)

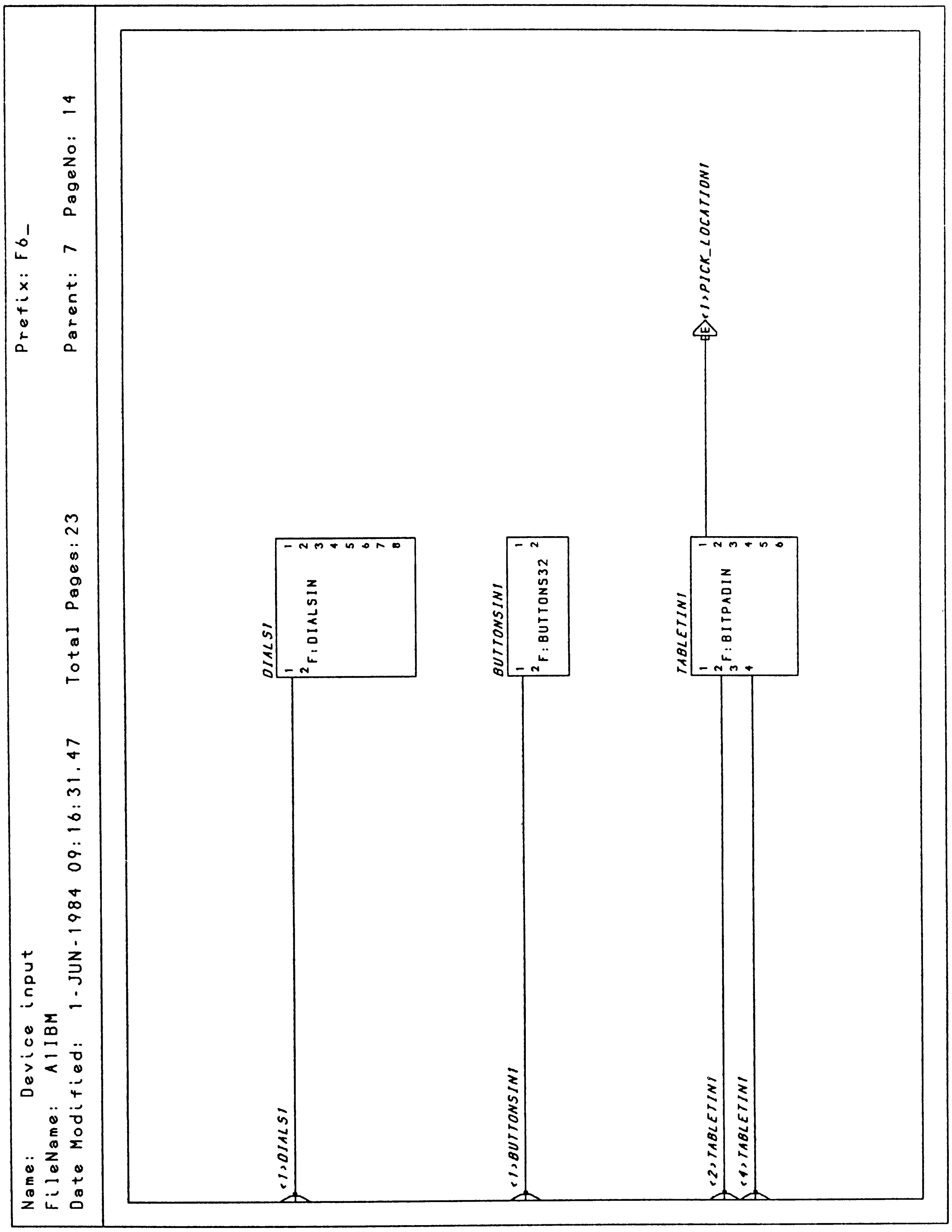


Figure 8-40. PS 390 Host Input Data Flow (IBM)

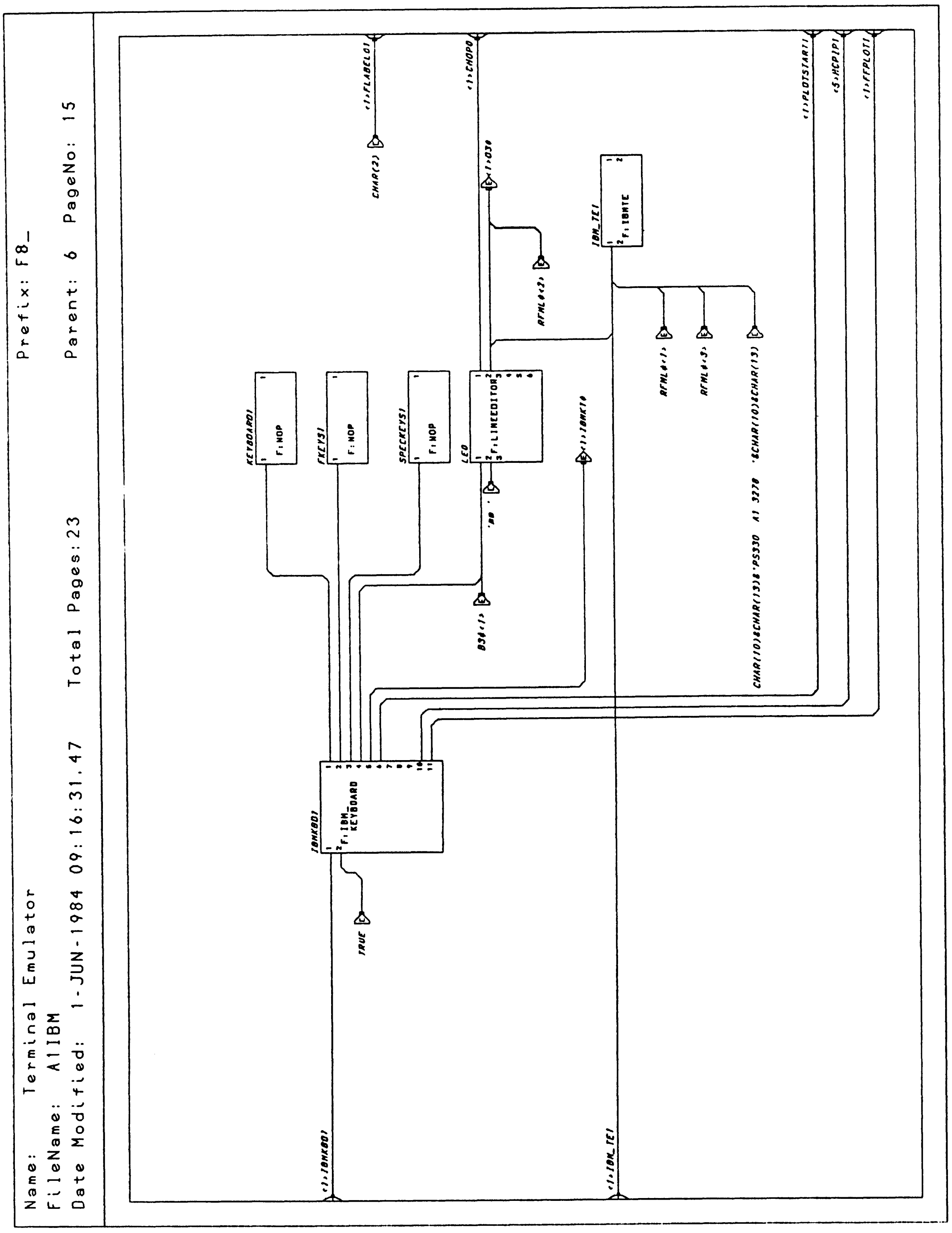


Figure 8-41. PS 390 Host Input Data Flow (IBM)

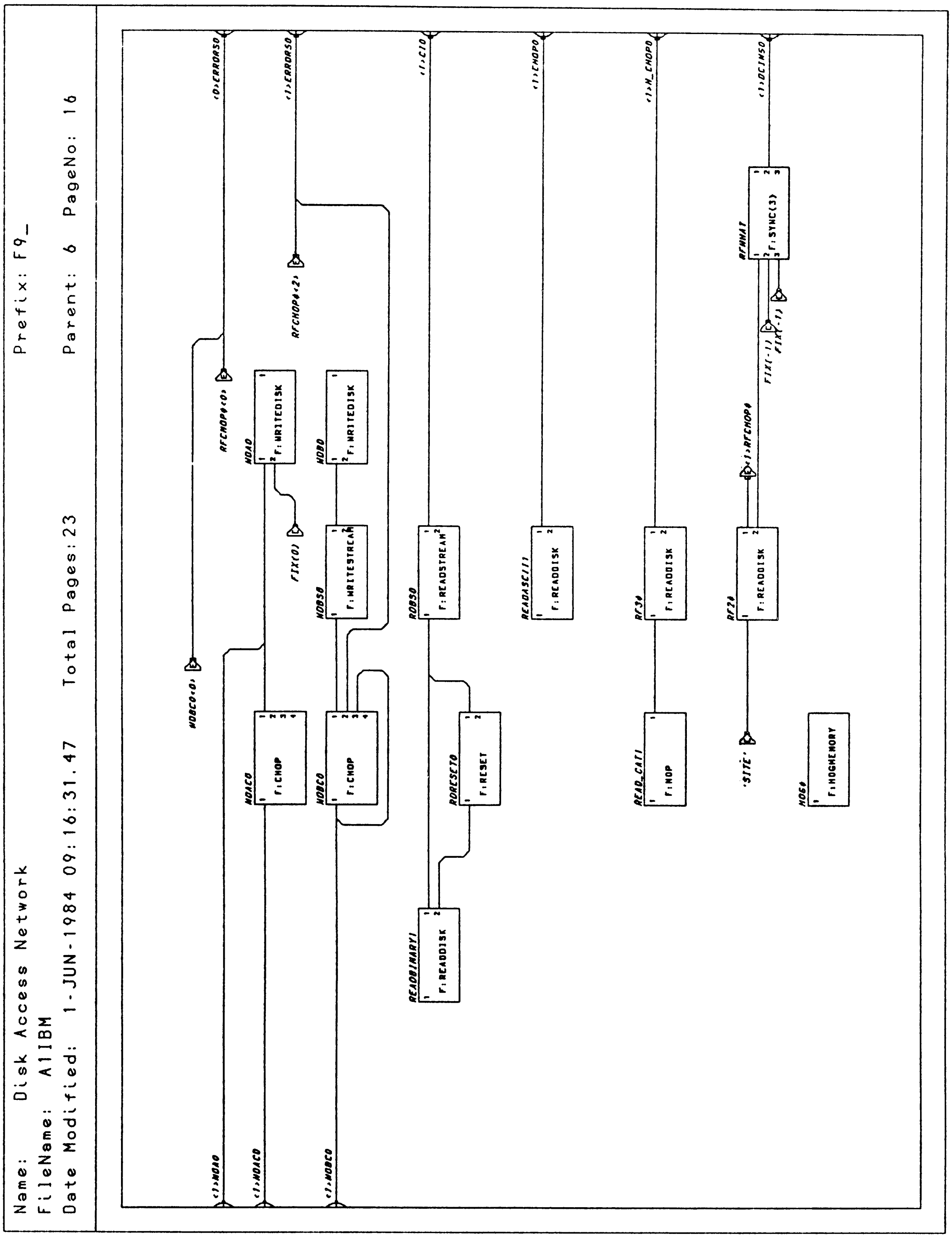


Figure 8-42. PS 390 Host Input Data Flow (IBM)

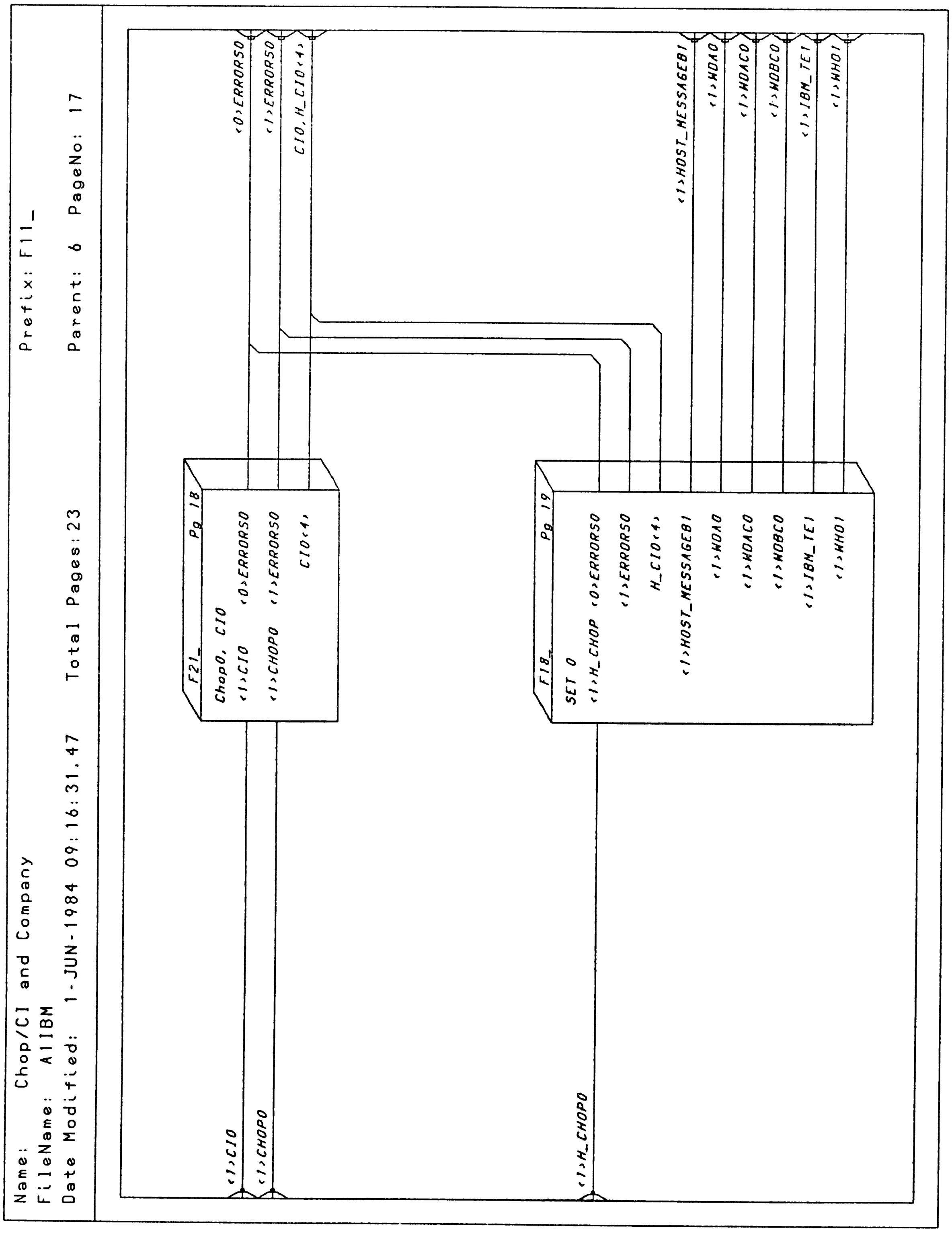


Figure 8-43. PS 390 Host Input Data Flow (IBM)

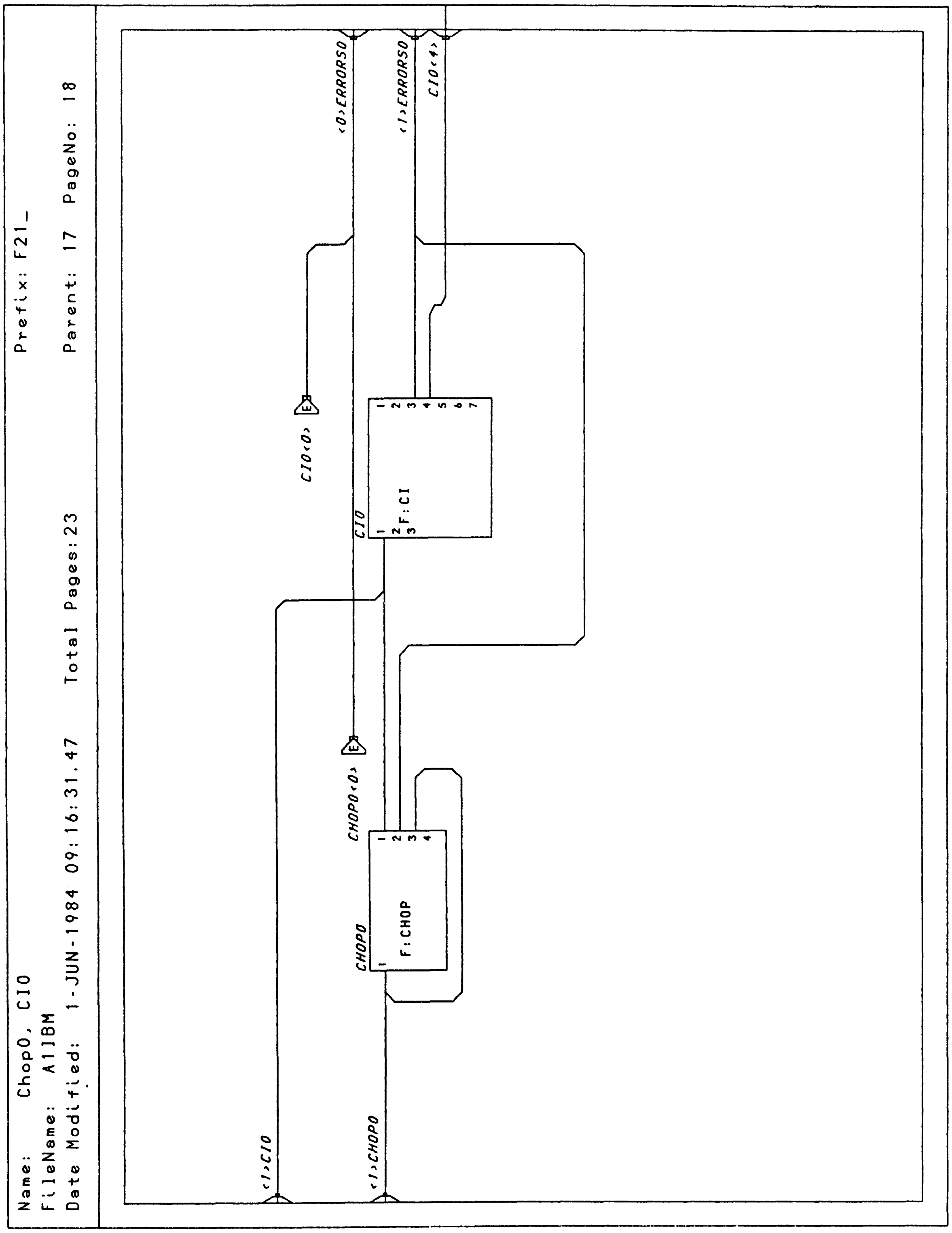


Figure 8-44. PS 390 Host Input Data Flow (IBM)

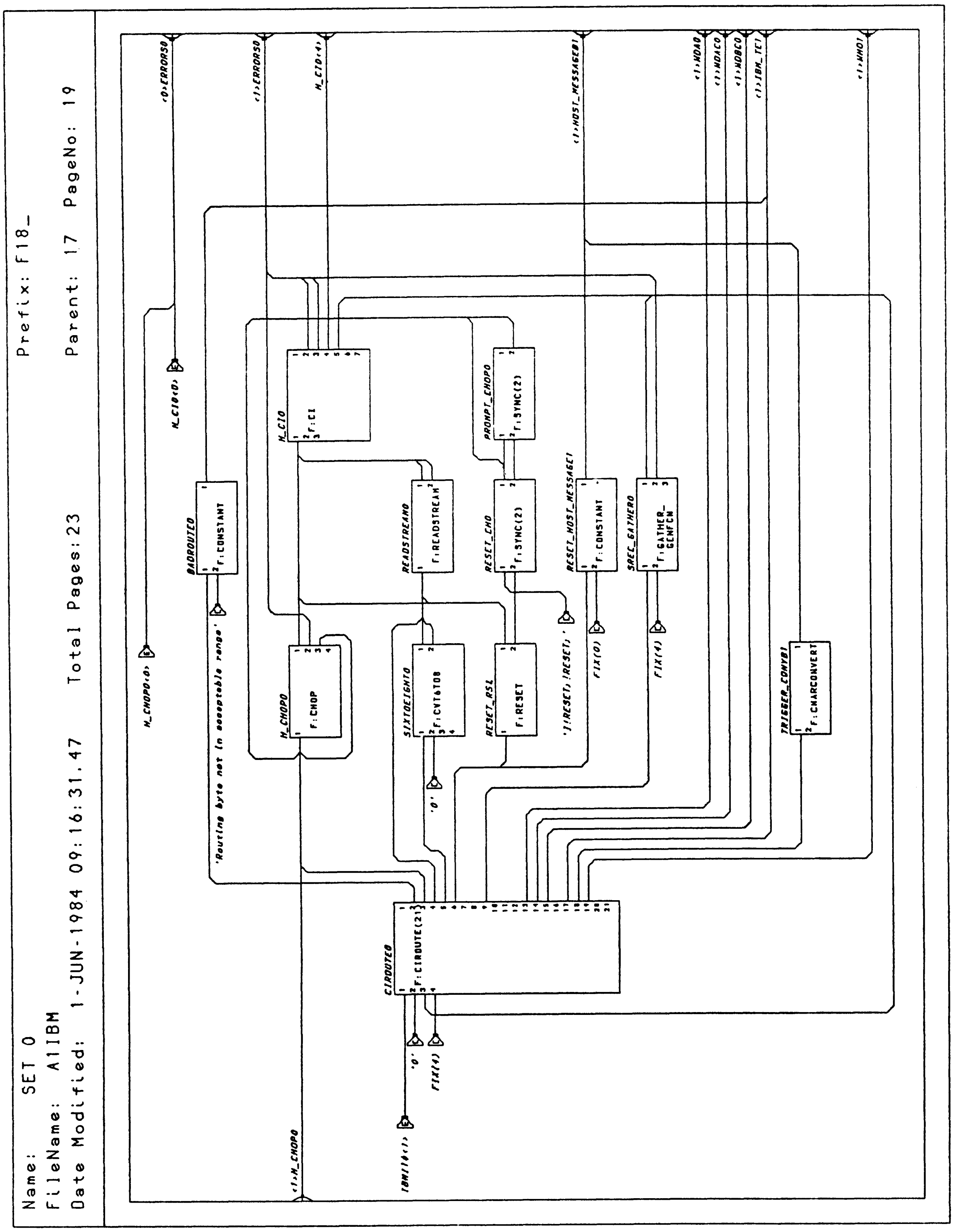


Figure 8-45. PS 390 Host Input Data Flow (IBM)

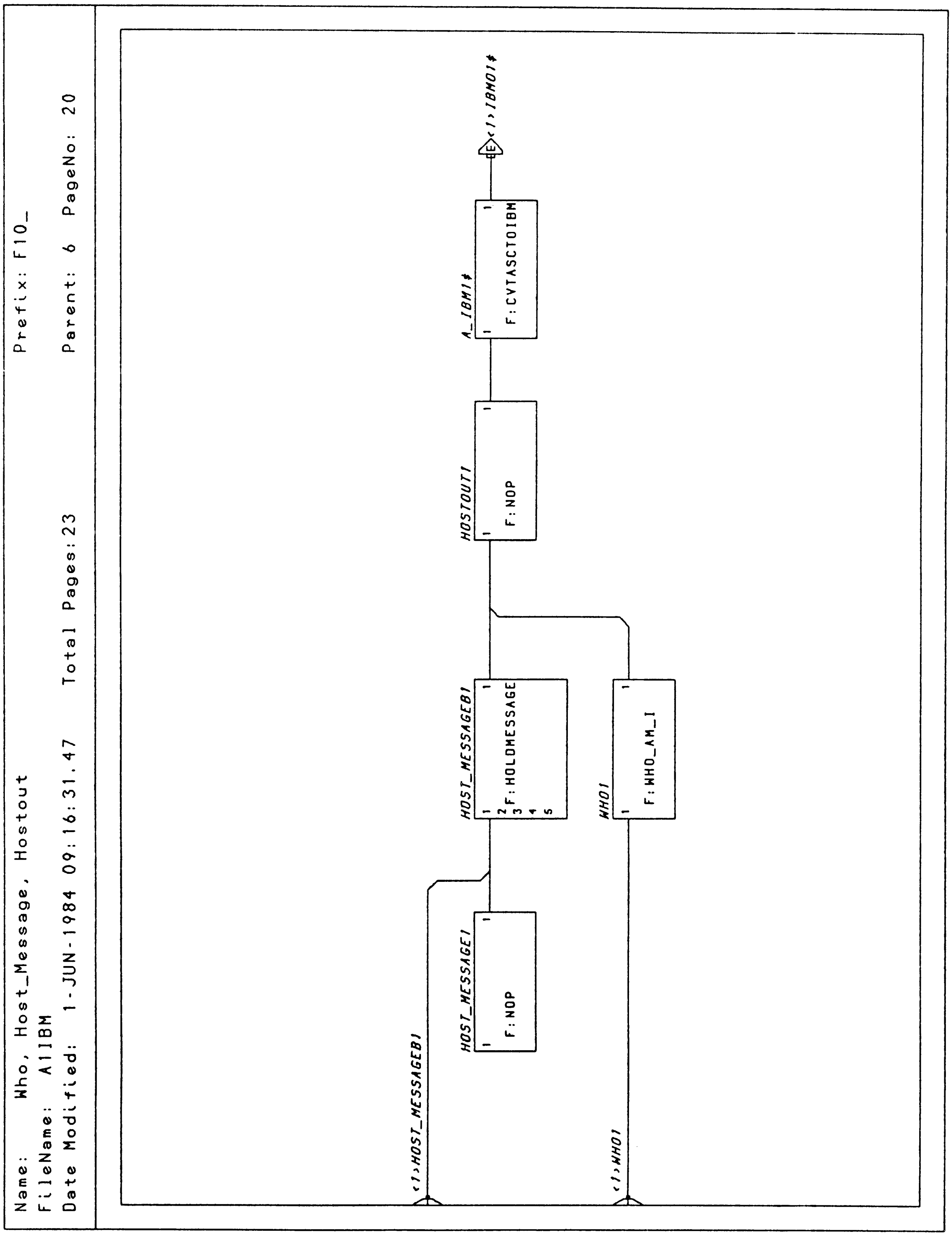


Figure 8-46. PS 390 Host Input Data Flow (IBM)

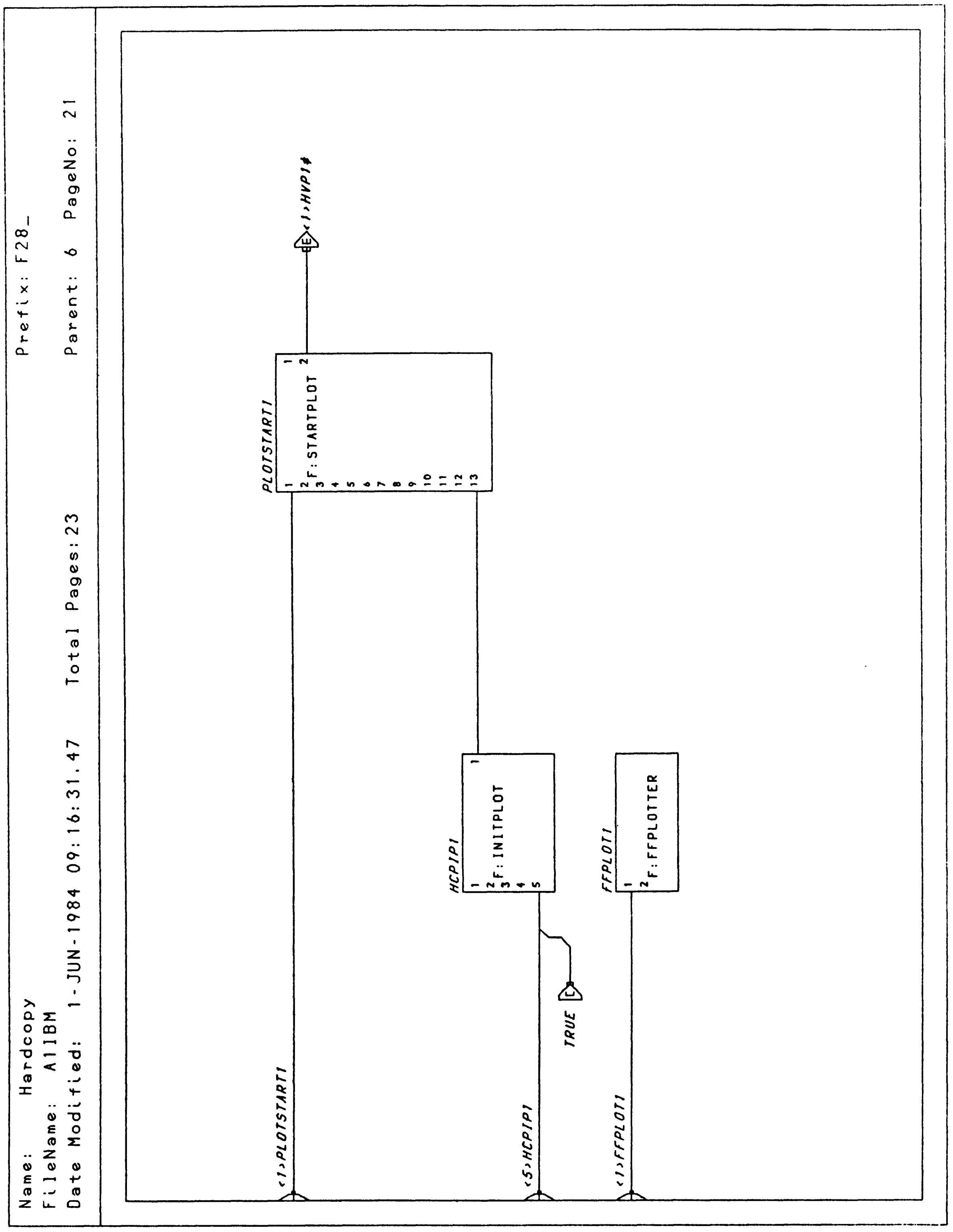


Figure 8-47. PS 390 Host Input Data Flow (IBM)

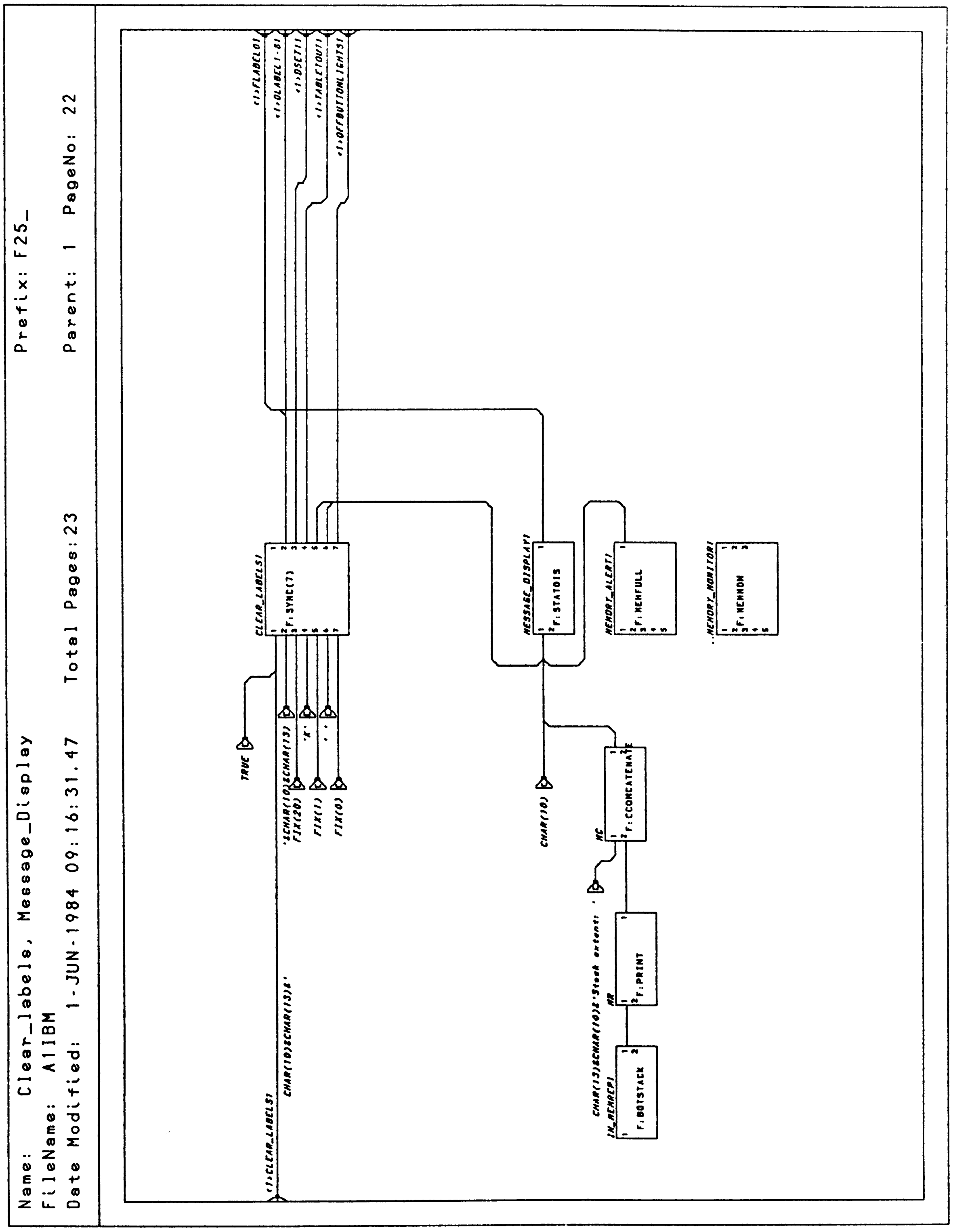


Figure 8-48. PS 390 Host Input Data Flow (IBM)

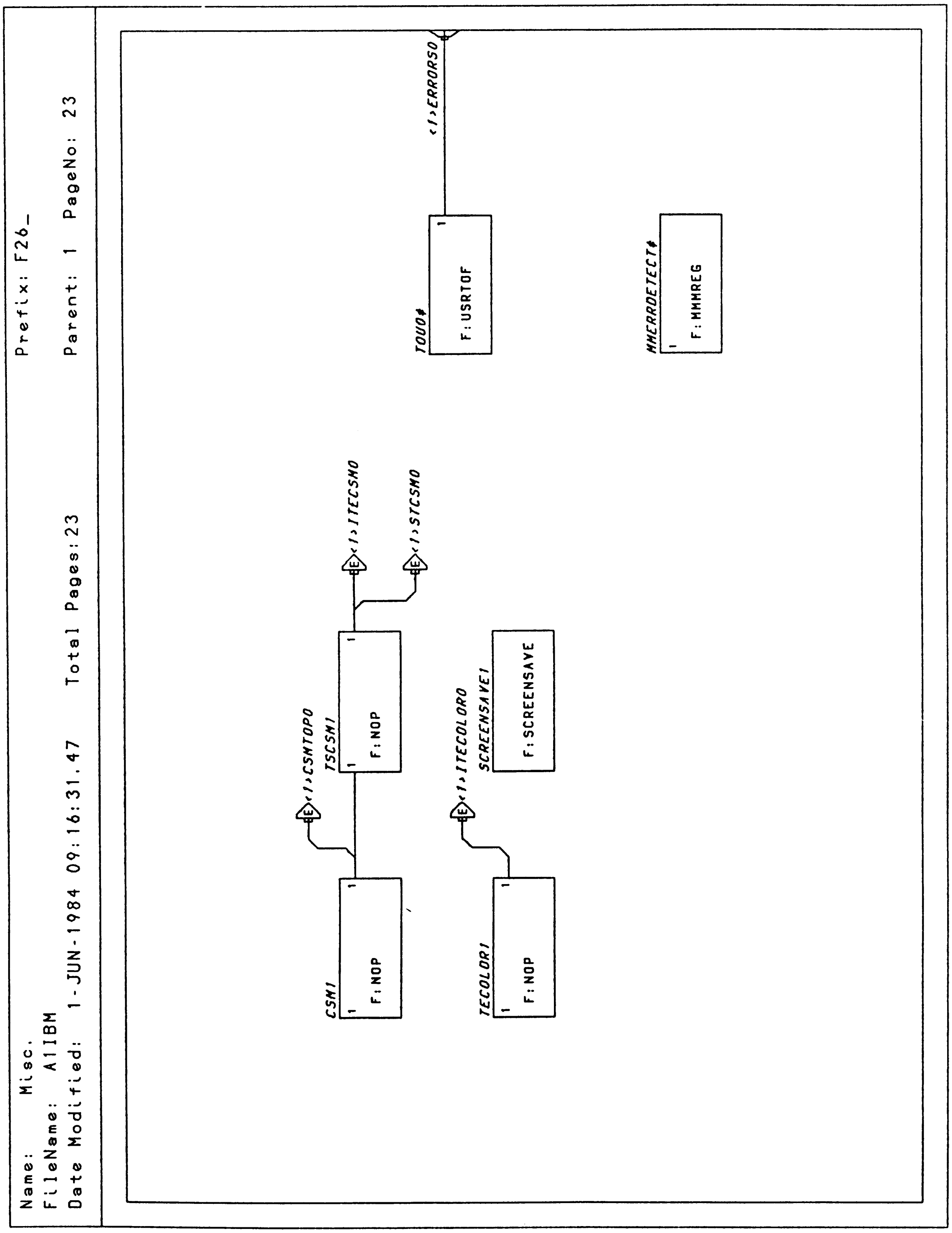


Figure 8-49. PS 390 Host Input Data Flow (IBM)

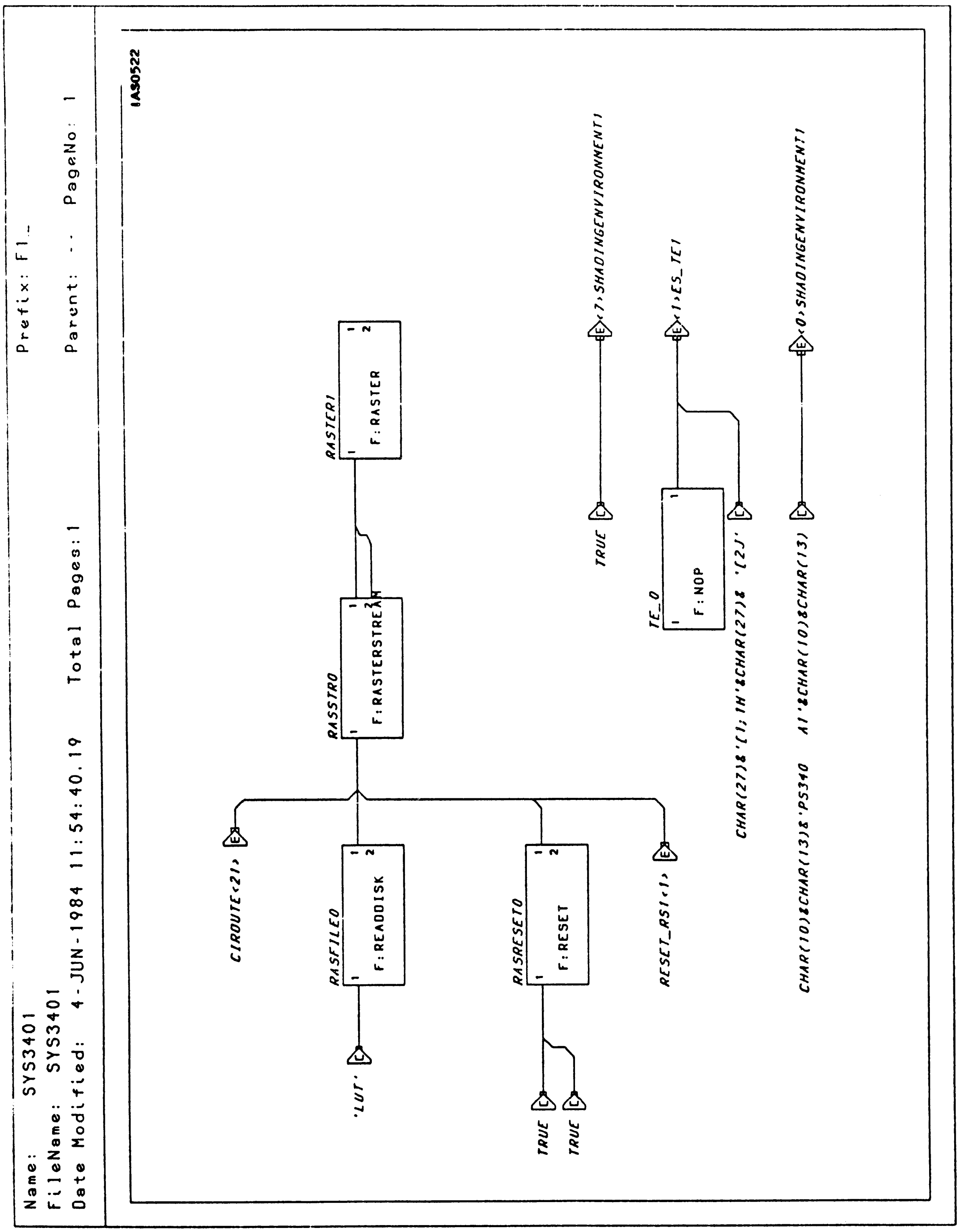


Figure 8-50. Raster System Host Input Data Flow (DEC)

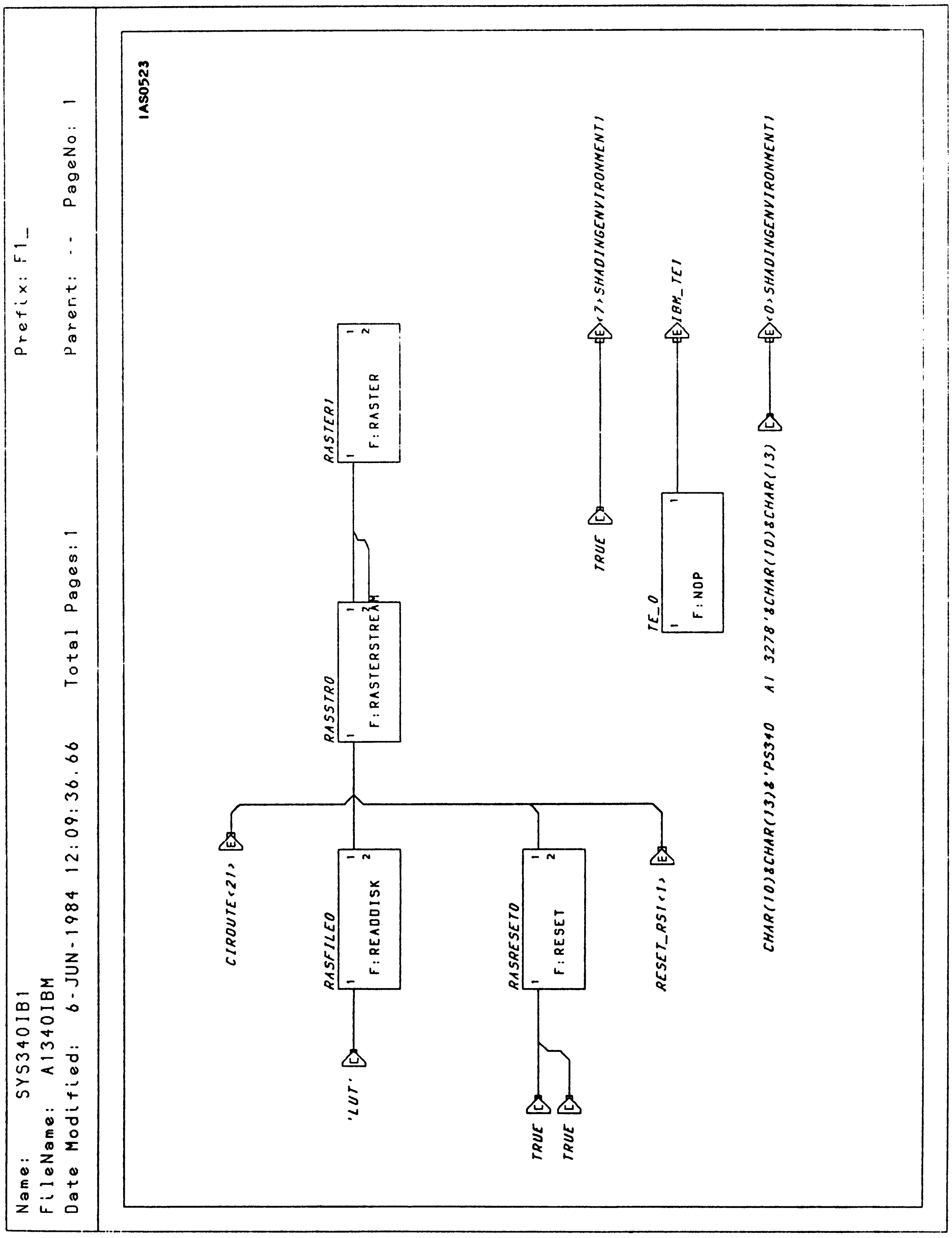


Figure 8-51. Raster System Host Input Data Flow (IBM)

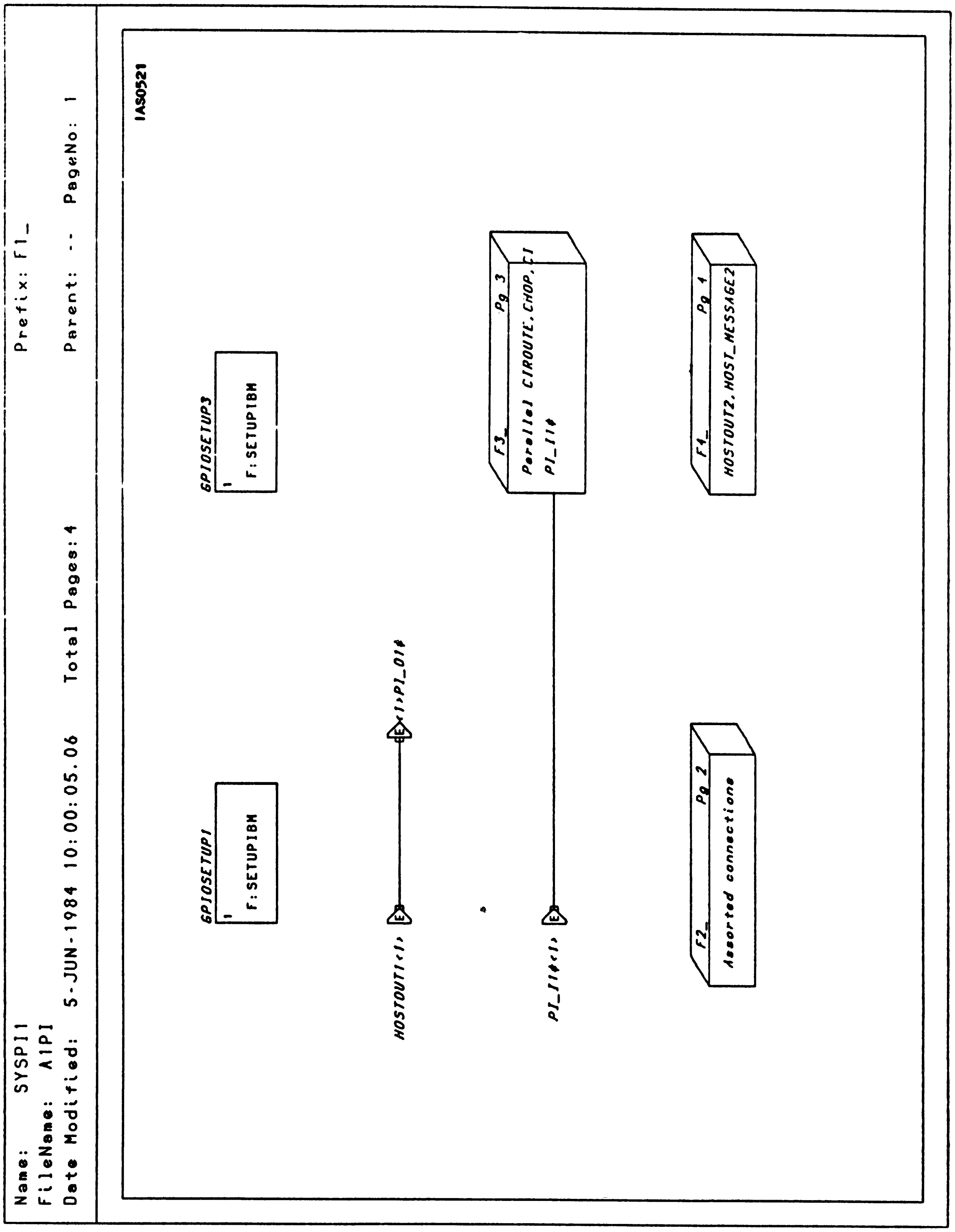


Figure 8-52. DEC Parallel Interface Host Input Data Flow

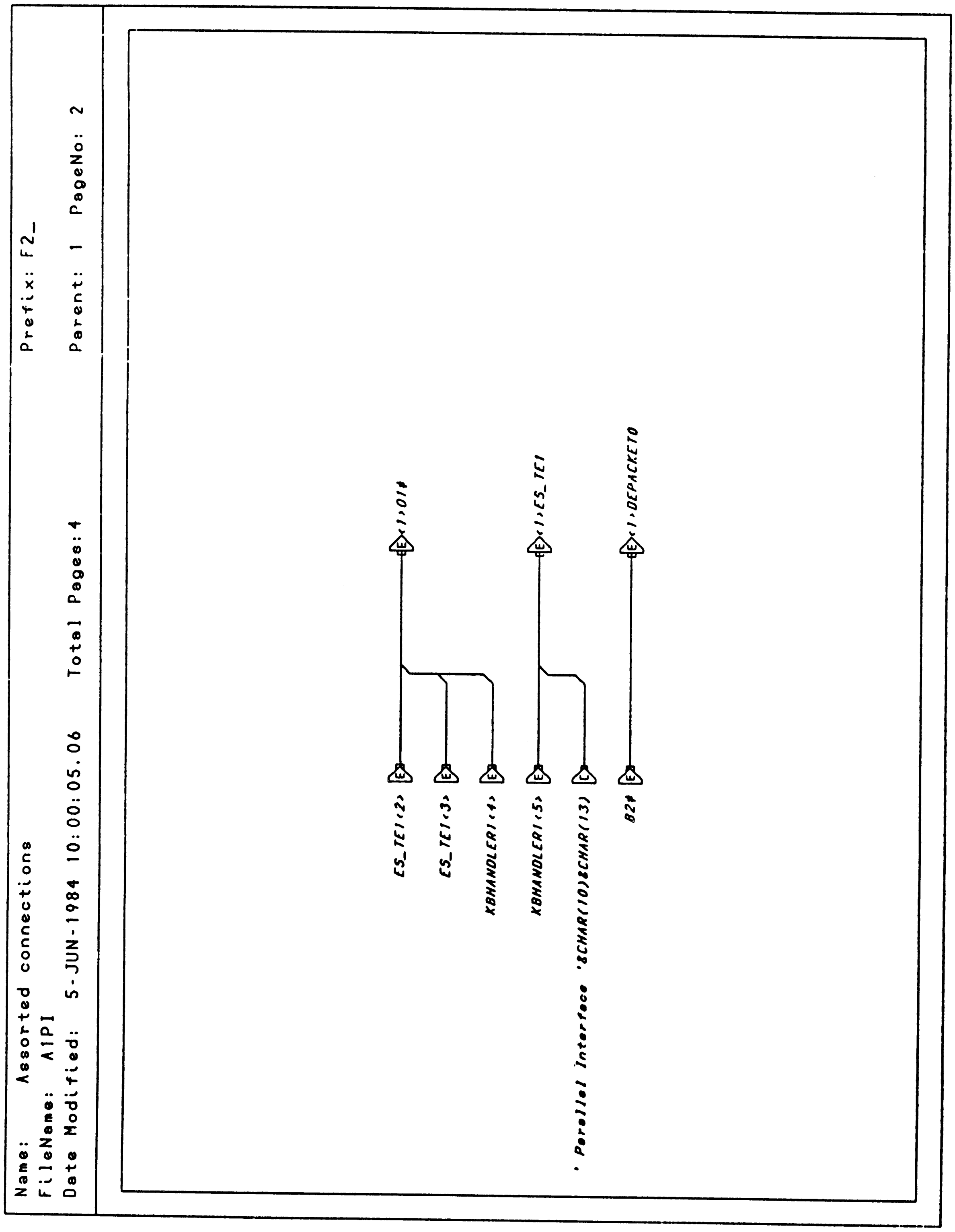


Figure 8-53. DEC Parallel Interface Host Input Data Flow

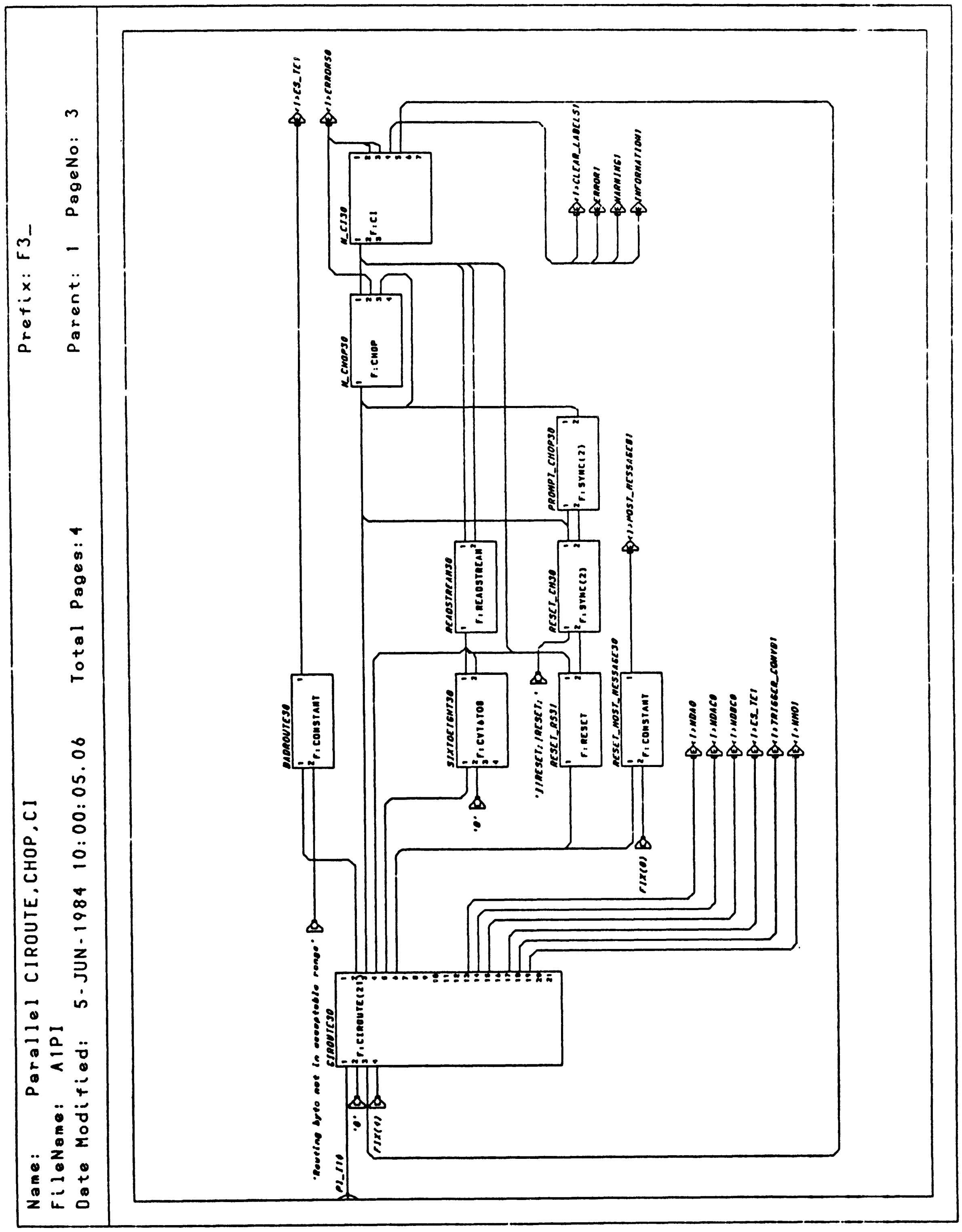


Figure 8-54. DEC Parallel Interface Host Input Data Flow

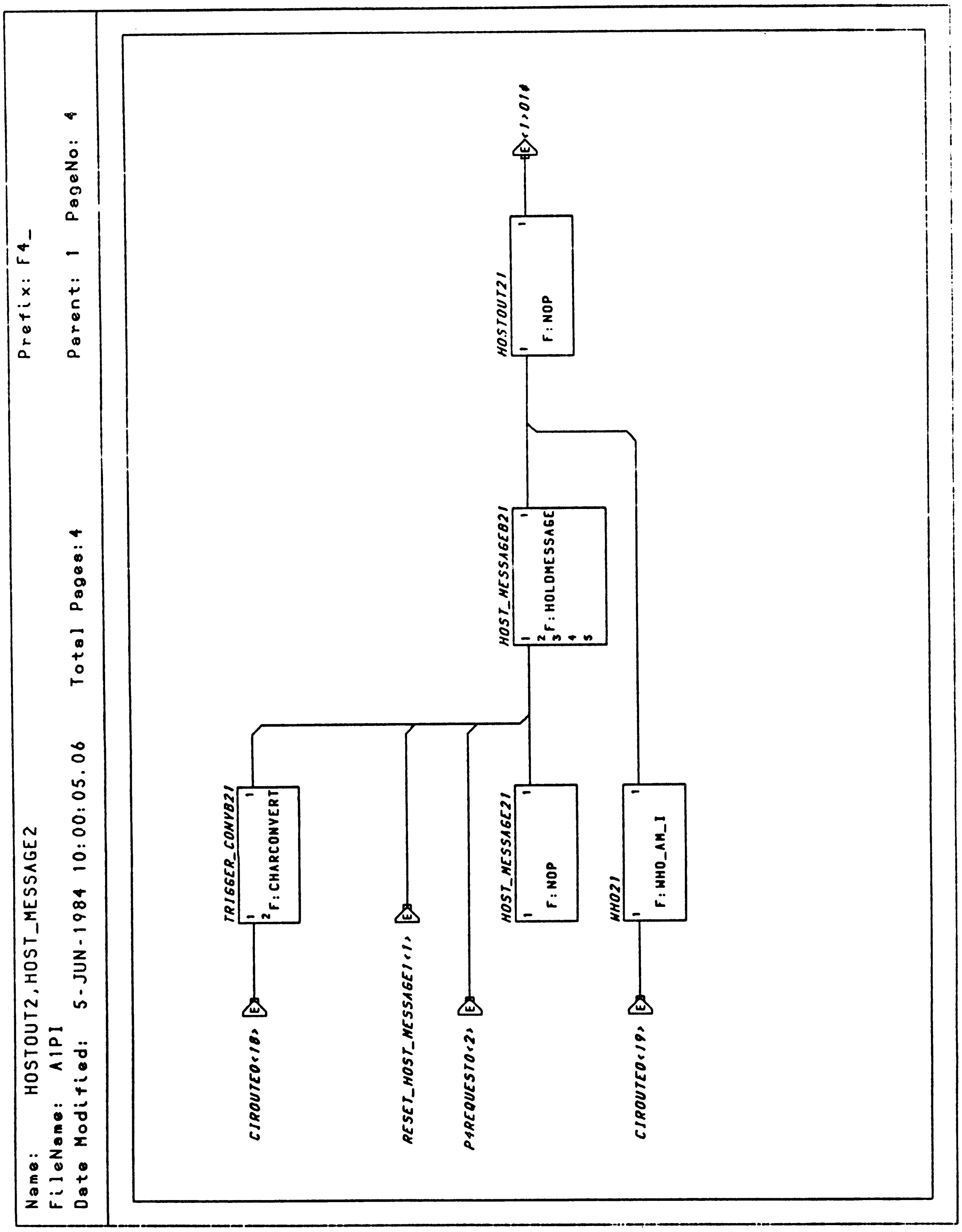


Figure 8-55. DEC Parallel Interface Host Input Data Flow

RM9. INITIAL STRUCTURES

CONTENTS

Section RM9
Initial Structures

This section discusses initial data structures and name suffixing including the system configure mode and its uses. The first section describes a runtime system and the initial data structures that are built by the PS 390 firmware. Following sections discuss the configure mode and name suffixing procedures.

Information for systems using DEC and IBM host computers is included. It is noted where the information for each configuration may be different.

1. Runtime System

The PS 390 is a runtime system, it does not act like a personal computer or provide a standard programming environment. The PS 390 does not have a file system, editor, compiler, or symbolic debugger. The runtime system is composed of PS 390 functions linked together to form the system function network. A PS 390 function can be viewed as self-contained program. With minor exceptions, it has no access to disk files, and deals with the world via messages and queues, one transaction at a time.

1.1 The Graphics Control Program

The Graphics Control Program is the collection of software that executes whenever the PS 390 is used as an interactive computer graphics terminal. The 68000 startup code loads the control program into local JCP memory.

The Graphics Control Program is made up of:

- Data structure definitions
- Scheduler
- Functions

1.1.1 Data Structure Definitions

The data structures are set up by Pascal procedures that define and make use of the following:

- Named entities; data structures that can be named and referenced.
- Alpha block; data structure that contains the location of a named entity.
- ACP state; contains the parameters the define the context of the display processor at any given time.

1.1.2 Scheduler

The PS 390 runtime system contains a scheduler that is activated once the initialization code on the firmware has been loaded. The scheduler loops to schedule and execute functions. When a function is instanced, it is assigned a default priority for execution. The scheduler uses this priority number to determine which active function will be scheduled next to be executed.

1.1.3 Functions

The PS 390 intrinsic system and user functions are described in Sections *RM2 Intrinsic Functions* and *RM3 Initial Function Instance*s.

1.2 Initial Data Structures

The initial data structures are built by the CONFIG.DAT file. These structures set up the framework that allow you to build displayable data structures. The initial data structures form the top nodes of a display structure that the JCP and ACP traverse to generate the display during each cycle.

1.3 Code for Initial Data Structures

The code that supports the initial data structure follows.

Initial Display Data Structure:

```
SCO$ := SET DISPLAYS ALL ON
   THEN VPF1$
VPF1$ := VIEW HORIZONTAL = -1:1 VERTICAL = -1:1 INTENSITY = 0:1
   THEN HVP1$;
HVP1$ := VIEW HORIZONTAL = -1:1 VERTICAL = -1:1 INTENSITY = 0:1
   THEN CSMTOPO
CSMTOPO := SET CSM OFF
   THEN GTO$
 SEND TRUE TO <-1>HVP1$
GTO$ := INSTANCE OF GVPO$, TVPO$, MDO$
```

Graphics Display Structure:

```
GVPO$ := VIEW HORIZONTAL = -1:1 VERTICAL = -1:1 INTENSITY = 0:1
   THEN PICK_LOCATION1;
PICK_LOCATION1: := SET PICK LOCATION = 0,0 .01, .01
   THEN GCURO$;
GCURO$ := INSTANCE WB$1, CT1$,
WB$! := WRITEBACK
   THEN GDO$;       {All Display Commands append to GDO$}
CT1$ := TRANSLATE BY 0,0,2
   THEN CURSOR1;
CURSOR1 := VECTOR_LIST ITEMIZED N=10
   p .035,.035 1 -.035,-.035 p -.035,.035 1 .035,-.035
   p .035,.035 1 -.035,-.035 p -.035,.035 1 .035,-.035;
```

Terminal Emulator Display Structure for DEC VT100:

```
TVPO$ := VIEW HORIZONTAL = -1:1 VERTICAL = -1:0
   THEN TENOSLAVEO$;
TENOSLAVEO$ := SET DISP ALL ON
   THEN TECSMO;
TECSMO := SET CSM OFF
   THEN TECOLORO;
TECOLORO := SET COLOR 240.0, 1.0
   THEN TDO$; {The Terminal Screen is appended to TDO$}
```

Terminal Emulator Display Structure for IBM 3278:

```
IVPO$ := CHAR FONT IMBFONT$
   THEN ITENOSLAVEO$;
ITENOSLAVEO$ := SET DISP ALL ON
   THEN ITECSMO;
 ITECSMO := SET CSM OFF
   THEN ITECOLORO;
ITECOLORO := SET COLOR 240, 1
   THEN IBMSCRO$;
IBMSCRO$ := INSTANCE IBMSCO$, IBMLINE$;
IBMLINE$ := VEC N = 2  -1, -.88  1, -.88;
```

Crash Message Display Structure:

```
CRASH_MSGS$:=BEGIN_STRUCTURE
    IF LEVEL =  17 THEN C17$;
    IF LEVEL =  15 THEN C16$;
    IF LEVEL =  16 THEN C16$;
    IF LEVEL >  17 THEN C16$;
    IF LEVEL <  0 THEN C16$;
    IF LEVEL =  0 THEN CO$;
    IF LEVEL =  1 THEN C1$;
    IF LEVEL =  2 THEN C2$;
    IF LEVEL =  3 THEN C3$;
    IF LEVEL =  4 THEN C4$;
    IF LEVEL =  5 THEN C5$;
    IF LEVEL =  6 THEN C6$;
    IF LEVEL =  8 THEN C8$;
    IF LEVEL =  9 THEN C9$;
    IF LEVEL =  10 THEN CA$;
    IF LEVEL =  11 THEN CB$;
    IF LEVEL = 12 THEN CC$;
    IF LEVEL = 13 THEN CD$;
    IF LEVEL = 14 THEN CE$;
END_STRUCTURE;
C16$:=CHAR 'Unknown crash';
CO$:=CHAR 'Mass memory Exhausted';
C1$:=CHAR 'OKINT/NOINT imbalance';
C2$:=CHAR 'Free block size invalid';
C3$:=CHAR 'Attempt to activate non-function or nil';
C4$:=CHAR 'NEW call in Nomemsched failed to find memory';
C5$:=CHAR 'Attempt to queue where fcn already waiting';
C6$:=CHAR 'Systemerror';
C7$:=CHAR 'TRAP7';
C8$:=CHAR 'Mass Memory Error';
C9$:=CHAR 'TRAP9';
```

```
CA$:=CHAR 'Multiple DISPOSE of same block';
CB$:=CHAR 'Block exponent not big enough';
CC$:=CHAR 'TRAP C';
CD$:=CHAR 'PASCAL Error';
CE$:=CHAR 'PASCAL Error';
C17$ := CHAR 'Unexpected exception';
```

Setup Mode Display Structure:

```
SVPO$ := VIEW HORIZONTAL= -1:1 VERTICAL = -1:1
    THEN SZO$;
STCSMO := SET CSM OFF
    THEN SSO$;
SSO$ := CHAR SCALE 0.03
    THEN SSO$;
SSO$ := INSTANCE OF
    S10$, S20$, S30$, S40$, S50$, S60$, S70$, S80$, S90$;
S10$ := CHAR -1,.9 'SETUP';
S20$ := CHAR -1,.8 ' ';
S30$ := CHAR -1,.7 'F2-SRM :T F3=Awrp:F F4=ANSI:T F5=VT52:F ';
S40$ := CHAR -1,.6 'F6=KPM :F F7=CKM :F F8=Cnum:T F9=Knum:T ';
S50$ := CHAR -1,.5 ' ';
S60$ := CHAR -1,.4 'F10= Define breakkey :^V       ';
S70$ := CHAR -1,.3 'F11= Move TE viewport, lower left corner ';
S80$ := CHAR -1,.2 'F12= Move TE viewport, upper right corner ';
S90$ := CHAR -1,.1 'Mode: TE  Term: On  Graf: On    ';
SAO$ := CHAR -1,.0 'Press special key to be breakkey, F1 to exit. ';
SBO$ := CHAR -1,.0 'Move corner with cursor keys, F1 to exit. ':
```

2. CONFIG.DAT

CONFIG.DAT is a file on one of the PS 390 diskettes. This file is read and processed during system boot. It contains commands to create the initial function instances and display structures. Before the CONFIG.DAT file builds any of the data structures, the system must first read the file. The firmware creates a simple function network that consists primarily of an instance of the F:READDISK and the F:CI(n) functions. The function network then reads the CONFIG.DAT file from the diskette. The command interpreter is in the privileged configure mode while reading the CONFIG.DAT file.

The command interpreter that processes the CONFIG.DAT file is separate and distinct from the command interpreter that handles user commands. These user command interpreters are initially in a non-privileged command mode.

3. Name Suffixing

Whenever you name anything or instance a function, the command interpreter assigns a specific suffix to that name, unless the command interpreter is in configure mode. The suffix is determined by the suffix that has been assigned to that instance of the command interpreter. Name suffixing is used to separate system level names and instances from user-originated names and instances.

In command mode, all suffixing is done by the command interpreter. However, in configure mode the command interpreter does not assign suffixes, so you are responsible for correctly suffixing any function or structure that is instanced when using system-level or user-level names.

The default suffix assignments for the PS 390 are as follows:

- 0 – suffix for system related functions. Names with this suffix are not directly accessible to the user outside of configure mode.
- 1 – suffix for user-defined and accessible names. All names with this suffix are accessible to the user.

If you are creating an instance of the command interpreter, you must name that instance with the correct suffix to assure the other functions created by this command interpreter will have the appropriate suffix. Only characters 0–7 are allowed as suffixes to the name of the command interpreter instance.

If the command interpreter used is suffixed with a 0 or a 1, it will suffix names that it creates with a 1. If it is suffixed with 2 or 3, it suffixes names it creates with a 3. If it is suffixed with 4 or 5, it suffixes names it creates with a 5. If it is suffixed with 6 or 7, it suffixes names it creates with a 7.

General system names are usually distinguished from all other names with the $ suffix.

NOTE

When the F:CI(n) function is instanced, the function creates PICK[suffix]. Therefore, the command interpreter should be created before downloading the remaining program, or given a suffix that will create the PICK[suffix] used in the program. If this is not done, all connections from PICK[suffix] that were made before instancing will be lost.

4. Using the Configure Mode

To access system-level functions, you must be able to access any name, regardless of the suffix. To do this, you enter the privileged configure mode. In this mode you have the capability of reconfiguring system functions. Use the following command to enter configure mode while in the normal mode of operation:

```
CONFIGURE password;
```

where **password** is the string defined by the setup password command (refer to Section *RM1 Command Summary*). If no password has been defined (the default case), any string can be entered.

Since the command interpreter is in configure mode, you must explicitly include suffixes on any names to affect a specific user. For example, if the SITE.DAT file (read from the diskette when the command interpreter is in configure mode) contains commands to send a site message to FLABEL0, the appropriate suffix is included at the end of the name and the commands in the SITE.DAT file appear as:

```
SEND 'E&S System 11, Site Manager - Scot Jones' to <1> FLABEL01;
```

RM10. TERMINAL EMULATOR

MODES AND FUNCTIONS

CONTENTS

TABLES AND FIGURES

Section RM10
Terminal Emulator
Modes and Functions

This section discusses the PS 390 terminal emulator for both the DEC VT100 and the IBM 3278 systems. Each terminal emulator is discussed from several perspectives. Sections 1, 2, and 3 will discuss VT100 terminal emulation, and section 4 will cover IBM 3278 terminal emulation.

Section 1 covers the ANSI modes and control sequences that are used to implement the DEC VT100 terminal emulation capabilities of the PS 390. Many of DEC's private sequences and modes for the VT100 are referred to in this section. More information on these sequences and modes is found in DEC's VT100 User Guide (EK–VT100–UG–002).

Section 2 covers the system functions that form the terminal emulator network and how data is received and passed between them.

Section 3 discusses the three communication modes of operation of the keyboard and how certain keys are translated within these modes. Operator information for the three communication modes used by the PS 390 keyboard is covered in Section *IS3 Operation and Communication.*

Section 4 discusses the PS 390 IBM 3278 terminal emulator. This section covers the system functions that form the terminal emulator network and how data is received and passed between them. The TE is also discussed in terms of the three communication modes of operation of the keyboard. Operator information for the three communication modes used by the PS 390 keyboard is covered in Section *IS3 Operation and Communication.*

Refer to the IBM publication, IBM 3270 Information Display System 3278 Display Station Operator's Guide (IBM #GA27–2890–3), for information on the use and operation of the PS 390/IBM terminal and keyboard.

The terminal emulator facility has characteristics and features that can be changed fairly easily by system programmers. Information for changing and adapting these features for both the DEC and IBM terminal emulators will be covered throughout the section.

1. ANSI Modes of Operation

The PS 390 operates under ANSI (and certain VT52) modes wherein it recognizes and responds to certain coded sequences whose syntax and semantics are in accordance with ANSI specifications. These modes determine how other coded sequences are to be interpreted and how the terminal will respond in certain situations.

Escape sequences are interpreted as control functions that set the mode of operation, (i.e. sending a particular escape sequence from the host to the terminal will determine whether the numeric keypad on the PS 390 keyboard generates the numeric value of the keycaps or the escape sequences that are used for EDT editing commands). The interpretation of the escape sequence is dependent on the mode in which the terminal is operating. The modes can be set or reset by sending escape sequences from the host to the terminal.

It is difficult to categorize the modes in a straightforward manner because some of them are dependent on the settings of other modes: if the ANSI (VT100) mode is set to FALSE (or OFF), then logically, the terminal will not be able to respond to any other ANSI control sequences. Some of these modes are standard to the DEC VT100, and some are specific to the PS 390. The modes and the escape sequences that can be used to set them will be discussed in later sections. The list below gives some idea of the modes and what they do.

- Send-Receive Mode (SRM (Local echo/Nolocal echo)) — determines whether keyboard input will be echoed to the display.
- ANSI Mode (DECANM) — determines whether the PS 390 will generate and respond to standard ANSI (VT100) escape sequences.
- VT52 Mode — allows the PS 390 to recognize VT52 coded sequences.
- Keypad Numeric Mode (DECKPNM) — causes the numeric keycap values to be sent from the numeric keypad to the host.
- Keypad Application Mode (DECKPAM) — causes the keys on the numeric keypad to transmit an escape sequence which begins with <ESC>O to the host.
- Cursor Key Mode (DECCKM) — enables the cursor keys to transmit the ANSI control sequences that cause the cursor movements indicated on the cursor keycaps.

The modes listed previously, as well as other modes that are specific to the PS 390, can be changed using the terminal emulator SETUP facility. A definition of these modes, their defaults, and how to change them is discussed in Section *IS3 Operation and Communication*.

1.1 Definition of Escape Sequences

An escape sequence is a sequence of characters that is used for control purposes to perform a control function and whose first character is the escape <ESC> (the ASCII X'1B') control character. Escape sequences are used to set and reset modes, as well as tell the terminal how to respond to coded sequences. These characters are not displayed as text on the screen, but instead cause the terminal to perform some action or change some internal parameter of operation. Control sequences are also used to change or define characteristics of the terminal. A control sequence is an escape sequence that provides supplementary controls and begins with the control sequence introducer (CSI). In VT100 emulation, the CSI is <ESC>[.

The sequences that the terminal emulator deals with take two general forms: those that may have parameters, and those that do not. Those not having parameters take the form <ESC>c, where c is a single character. Those that may have parameters take the form:

```
<ESC>[P1;P2;...Pnc
```

where:

<ESC>[is the control sequence introducer.

P1....Pn are the parameters (none need be present).

; is used to separate parameters.

c is the final character that determines which control sequence is being defined.

The parameters are numbers expressed in their ASCII form. In sequences that use private or non-standard parameters, the first character of the parameter string is "?" for DEC private sequences and ">" for E&S private sequences.

1.2 SET and RESET - SM, RM

The SET and RESET control sequences are used to set and reset certain modes of the terminal. These control sequences for setting or resetting these modes are sent from the host. The modes that can be set or reset are listed below, along with the set and reset escape sequences.

- Send-Receive Mode (SRM)
- ANSI-VT52 (DECANM)
- Cursor Key Mode (DECCKM)
- E&S private sequences

The SET and RESET control sequences are:

```
SM:  <ESC>[Pnh

RM:  <ESC>[Pnl
```

where *n* is the parameter that determines which mode is to be set, i.e.,

```
<ESC>[?1h
```

would set the Cursor Key mode (DECCKM).

1.3 Send-Receive Mode (SRM) - Local Echo/Nolocal Echo

The SRM mode can be set or reset from the host by sending the proper control sequence, by using the SETUP facility of the terminal emulator package, or by including the appropriate ASCII characters in the SITE.DAT file. (Refer to Section *IS3 Operation and Communication* for SETUP, or to section 3.2 of this guide for information on the SITE.DAT file.) This mode determines whether the screen receives the input from the keyboard on the host line, or from a PS 390 system function. If the host line is half duplex, the host does not echo the keys as they are sent from the TE to the host. This mode must be reset so that the characters that are received by the TE from the keyboard will be displayed on the screen.

If the line to the host is full-duplex, the host retransmits the keys it receives from the keyboard back to the terminal, and they are then displayed on the screen. In this case, SRM should be set so that the characters will not appear on the screen twice: once as they are keyed in, and once as they are received back from the host.

1.4 Send-Receive Mode (SRM) Escape Sequences

"12" is the parameter that designates SRM.

<ESC>[12h	SET SRM. Do not send keyboard input to the display.
<ESC>[12*l*	RESET SRM. Send keyboard input to the display, with [CR] (Carriage Return) displaying as [CRLF] (Carriage Return-Line Feed).

1.5 ANSI - VT52 Mode Escape Sequences

The ANSI-VT52 modes can be set or reset with the (SM/RM) control sequences. The VT52 set state causes VT52 compatible escape sequences to be interpreted and executed. The ANSI set state causes only ANSI (VT100) compatible escape sequences to be interpreted and executed. The ANSI-VT52 modes are private, using a private string parameter. The first character in the string must be "?", with "2" designating ANSI-VT52 mode. The recognition of VT52 sequences may be turned off by using the **<ESC><** sequence when in the VT52 mode.

<ESC>[?2h	SET ANSI mode. Escape sequences will be interpreted as ANSI; keys will be translated accordingly.
<ESC>[?2*l*	SET VT52 mode. Escape sequences will be interpreted as VT52; keys will be translated accordingly.

1.6 Directional Cursor Keys - (DECCKM)

The four directional cursor keys of the keyboard have a single mode that may be set or reset using the SM/RM control sequences. The Cursor Key mode is similar to DEC's DECCKM. When this mode is reset (the default at power-up), the cursor keys transmit the ANSI control sequences that cause cursor movement as indicated by the arrows on the keycaps. When the Cursor Key mode is set, the keys are in an application mode, and like the numeric keypad, transmit escape sequences.

When the VT52 mode is in effect, the sequences have no intermediate characters, and are the same regardless of the setting of the Cursor Key mode. The following table shows what is transmitted in the Reset (RM) and Set (SM) modes.

Table 10-1. CURSOR KEY Transmission

CURSOR KEY	VT52 (SET - MODE (RESET)	ANSI MODE RESET MODE	ANSI MODE SET MODE
Up	<ESC>A	<ESC>[A	<ESC>OA
Down	<ESC>B	<ESC>[B	<ESC>OB
Right	<ESC>C	<ESC>[C	<ESC>OC
Left	<ESC>D	<ESC>[D	<ESC>OD

1.7 Cursor Key Mode Escape Sequences

The Cursor Key mode is also a private mode and uses the private parameter string. The first character in the string must be "?". "1" is the parameter that designates Cursor Key mode.

<ESC>[?1h	SET Cursor Key Mode. Cursor keys will now cause <ESC>Oc sequences to be sent.
<ESC>[?1*l*	RESET Cursor Key Mode. Cursor keys will now cause the <ESC>[c sequences to be sent.

1.8 E&S Private ANSI Commands for Function Keys, Numeric Keypad and Cursor Keys

The Function keys, the numeric keypad, and the cursor keys can be placed under the control of the user-application program. The modes to do so may be set or reset using E&S private ANSI commands.

For example, when Fkeys Always is set, the output of the Function Buttons is always sent to FKEYS<1>. When the mode controlling the routing of the output of the numeric keypad or the cursor keys is set, the numeric value (as input to function networks) of these keys are available to PS 390 function networks in any of the three communication modes (Terminal Emulator, Command, or Local). These keys (numeric keypad or cursor) cause integers to appear at output<9> of KBhandler. (Note: KBhandler is an instance of the function F:K2ANSI.) When reset, the key values will be sent through KBhandler to the host, the command interpreter, SPECKEYS, etc., depending on the communication mode of the keyboard.

Multiple parameters are allowed per command, i.e., <ESC>[>10;11;12h would cause all function keys, cursor keys, and keypad keys to go to the user application.

ANSI SEQUENCES	DESCRIPTION
<ESC>[>2h	Set no/local echo. The TE will not locally echo keys. When reset, the TE locally echoes keys.
<ESC>[>3h or <ESC>[>1h	Set auto-wrap. The TE adds <CRLF> if it receives more than 80 characters without getting <CRLF>. When reset, the TE puts additional characters in column 80, overwriting the last one.
<ESC>[>4h	Set ANSI. The TE recognizes ANSI control sequences. When reset, the TE responds like a teletype terminal. When the reset sequence is sent from the host to the PS 390, all further ANSI commands are ignored (including <ESC>[>4l).
<ESC>[>5h	Set VT52. The TE will recognize VT52 control sequences. When reset, the TE will not recognize VT52 control sequences.
<ESC>[>6h	Set KPM. The numeric keypad sends control sequences. When reset, the numeric keypad sends numbers.
<ESC>[>7h	Set CKM. The cursor keys send control sequences. When reset, the cursor keys send cursor control sequences.
<ESC>[>8h	Set Cnum. The numeric keypad sends numbers in CI mode. When reset, the numeric keypad sends ↑Vc in CI mode.
<ESC>[>9h	Set Knum. The numeric keypad sends numbers in KB mode. When reset, the numeric keypad sends ↑Vc in KB mode.

<u>ANSI SEQUENCES</u>	<u>DESCRIPTION</u>
<ESC>[>10h	Set Fkeys Always. Except in TE SETUP, the numeric value of the Function Keys will always appear at FKEYS<1>, regardless of the PS 390 communication mode. When reset, the Function keys become VT100 keypad keys.
<ESC>[>11h	Set Cursor Keys Always. Except in TE SETUP, the numeric value of the cursor keys will always appear at KBhandler<9> regardless of the PS 390 communication mode.
<ESC>[>12h	Set Keypad Keys Always. Except in TE SETUP, the numeric value of the numeric keypad keys will always appear at KBhandler<9> regardless of the PS 390 communication mode.

There are four other E&S private set and reset escape sequences that can be used to set display features of the PS 390. These escape sequences change the status of the displays affected by the TERM and GRAPH keys.

<u>ANSI SEQUENCES</u>	<u>DESCRIPTION</u>
<ESC>[>13h	Turns the TE display ON.
<ESC>[>13*l*	Turns the TE display OFF.
<ESC>[>14h	Turns the GRAPH display ON.
<ESC>[>14*l*	Turns the GRAPH display OFF.
<ESC>[>10*l*, etc.	Reset the various modes. When reset, the keys function under the modes in effect. (For example, if "Fkeys always" is reset, and DECKPAM is set, the last four Function Keys will generate control sequences used by DEC's EDT and KED editing programs.

Any of the above modes may be set or reset by entering the appropriate characters in the SITE.DAT file, or by sending the appropriate sequence to <1>ES_TE1.

1.9 Values Appearing at KBhandler<9>:

When Keypad Keys Always is set, the numeric keypad keys pass their own value (except 0). For instance, pressing the 5 key in Keypad Keys Always mode causes an integer 5 to be output from KBhandler<9>. The remaining keys spiral out from the "9" key:

```
'-'    is 10
','    is 11
ENTER  is 12
'.'    is 13
'0'    is 14
```

Cursor keys:

```
Up cursor is 15
Down cursor is 16
Left cursor is 17
Right cursor is 18
```

These modes may be set or reset by entering the appropriate ASCII characters in the SITE.DAT file. For example:

```
SEND CHAR(27) & '[>10h' to <1>ES_TE1;
```

would set the Fkeys Always mode.

1.10 Numeric Keypad - (DECKPNM and DECKPAM)

The characters or sequences transmitted by the numeric keypad are dependent on a number of modes and configurations that can be set by the programmer. Normally, the numeric keypad transmits the codes shown on the key caps. However, in some host applications (DEC's editor utilities EDT and KED), these keys need to be interpreted as program function keys to cause some action to take place.

To differentiate these keys from the number and character keys on the main keyboard, the numeric keypad has two modes; a keypad numeric mode, and a keypad application mode. In the application mode, the keys transmit specific sequences. (Refer to Table 10-2 and Table 10-3.)

1.11 Numeric Keypad Escape Sequences

The keypad modes are set up by sending two different escape sequences from the host and eventually to the terminal emulator network (KBhandler).

```
<ESC>>
```

causes the keycap values (numeric and other) to be sent to the host when the keys are pressed. This is the keypad numeric mode that corresponds to DEC's DECKPNM.

```
<ESC>=
```

puts the keypad in the keypad application mode (DEC's DECKPAM). In this mode, pressing the keys causes them to transmit an escape sequence that begins with <ESC>O.

Setting the DEC VT52 mode (as opposed to the ANSI mode) will also affect the translation of these keys. Table 10-2 shows what is transmitted in the two modes when ANSI is set. Table 10-3 shows what is transmitted in the two modes when VT52 is set.

Table 10-2. Keypad Transmissions in ANSI Mode

	KEY CAP	NUMERIC MODE (DECKPNM)	APPLICATION MODE (DECKPAM)
	0	0	<ESC>Op
	1	1	<ESC>Oq
	2	2	<ESC>Or
	3	3	<ESC>Os
	4	4	<ESC>Ot
	5	5	<ESC>Ou
	6	6	<ESC>Ov
	7	7	<ESC>Ow
	8	9	<ESC>Oy
	–	–	<ESC>Om
	,	,	<ESC>O*l*
	.		<ESC>On
	ENTER	[CR]	<ESC>OM
**	P1(F9)	<ESC>OP	<ESC>OP
**	P2(F10)	<ESC>OQ	<ESC>OQ
**	P3(F11)	<ESC>OR	<ESC>OR
**	P4(F11)	<ESC>OS	<ESC>OS

Table 10-3. *Keypad Transmissions in VT52 Mode*

	KEY CAP	NUMERIC MODE (DECKPNM)	APPLICATION MODE (DECKPAM)
	0	0	<ESC>?p
	1	1	<ESC>?q
	2	2	<ESC>?r
	3	3	<ESC>?s
	4	4	<ESC>?t
	5	5	<ESC>?u
	6	6	<ESC>?v
	7	7	<ESC>?w
	8	9	<ESC>?y
	–	–	<ESC>?m
	,	,	<ESC>?*l*
	.		<ESC>?n
	ENTER	[CR]	<ESC>?M
**	P1(F9)	<ESC>P	<ESC>P
**	P2(F10)	<ESC>Q	<ESC>Q
**	P3(F11)	<ESC>R	<ESC>R
**	P4(F12)	<ESC>S	<ESC>S

1.12 Escape Sequences that Affect Screen Display

There are a number of escape sequences that can be sent from the host causing some action to take place in the terminal that affect the screen display. These include cursor position, scrolling, deletion of text, scrolling regions, and selective graphic rendition.

The following sections describe the escape sequence commands that implement these actions.

1.13 Cursor Movement Command Escape Sequences

The cursor movement commands UP, DOWN, FORWARD and BACK are identical in form except for the final character. They take the form

```
<ESC>[Pc
```

where *P* is the number of positions to move, and *c* is A for UP, B for DOWN, C for FORWARD, and D for BACK. If *P* is 0, 1, or absent, it is interpreted to be 1.

These sequences, with *P* absent, are generated by the cursor keys when the Cursor Key mode is reset.

If a given cursor command causes the cursor to move out of the display area, the cursor is set at the edge of the display area in the direction of the move. Scrolling does not take place. If the cursor were on the bottom line, and the TE received <ESC>[26B, nothing would happen. The cursor would remain on the bottom line, and no scrolling would take place.

```
<ESC>[PA    CUU - Move the cursor P lines upward.

<ESC>[PB    CUD - Move the cursor P lines down.

<ESC>[PC    CUF - Move the cursor P columns forward.

<ESC>[PD    CUB - Move the cursor P columns back (left).
```

The cursor position and horizontal vertical position (CUP, HVP) commands take the same form except for the final character. They take the form

```
<ESC>[Pl;PcC
```

where *Pl* is the line number to move to, *Pc* is the column to move to, and *C* is "H" for CUP and "f" for HVP (VT100 editor function). If one of these commands would cause the cursor to move out of the display area, it is set at the edge of the display area in the direction of the move. Scrolling does not take place. With no parameters present, it is equivalent to a cursor to home action.

```
<ESC>[Pl;PcH    CUP - Move cursor to line Pl, column Pc.

<ESC>[Pl;Pcf    HVP - Move cursor to line Pl, column Pc.
```

1.14 Index, Next Line, and Reverse Index Command Escape Sequences (IND, NEL, RI)

These commands move the cursor, but may also cause scrolling to occur. All of them take the form <ESC>c.

```
<ESC>D   IND - Move the cursor down one line, maintaining column
         positioning. If the TE is at the bottom line of the
         scrolling window when IND is received, a scroll-up is
         performed.
```

`<ESC>E`	NEL - Move the cursor down one line and to column 1. If the TE is at the bottom line of the scrolling window when NEL is received, a scroll-up is performed.
`<ESC>M`	RI - Move the cursor up one line, maintaining column position. If the TE is at the top line of the scrolling window when RI is received, a scroll-down is performed.

1.15 Erase Commands Escape Sequences (ED, EL)

The Erase in Display (ED) command takes the form

```
<ESC>[PJ
```

where P selects a specific erasing action. If P is absent, it is interpreted to be 0.

`<ESC>[J` or `<ESC>[OJ`	Erase the display from the cursor to the end of the screen.
`<ESC>[1J`	Erase the display from the beginning of the screen to the cursor.
`<ESC>[2J`	Erase the entire screen.

The Erase in Line (EL) command takes the form:

```
<ESC>[PK
```

where *P* selects a specific erasing action. If *P* is absent, it is interpreted to be 0.

`<ESC>[K` or `<ESC>[OK`	Erase from the cursor to the end of the line.
`<ESC>[1K`	Erase from the beginning of the line to the cursor.
`<ESC>[2K`	Erase the entire line.

1.16 Set Top and Bottom Margins Command Escape Sequence (DECSTBM)

This command allows a scrolling window to be defined. Inside the given scrolling window, the lines scroll as they normally would for the entire screen. Outside of the window, lines do not scroll. This command also causes the cursor to be positioned in the upper-left corner of the scrolling region as defined.

The form of this command is:

```
<ESC>[Pt;Pbr
```

where *Pt* is the top line of the scrolling window, *Pb* is the last line of the scrolling window, and *r* designates this command.

This command also requires that *Pt* < *Pb* since the scrolling window must be logical and contain a minimum of two lines. Should an illegal set of parameters be defined, the current setting of the window remains unchanged. For example, the sequence

```
<ESC>[Pt;Pbr
```

would make the scrolling window *Pt* to *Pb*, inclusive.

1.17 Set Graphic Rendition Command Escape Sequences (SGR)

The intent of this command is to make some part of the text displayed on the screen stand out, in contrast to the rest of the screen. The form of the command is

```
<ESC>[Pm
```

where *P* selects some form of graphic rendition. If *P* is absent or 0, then all forms of graphic rendition are turned off. As the most common methods used to make the contrast are difficult or expensive to implement on the PS 390, the command is interpreted by underscoring the selected text in the display.

```
<ESC>[Pm      (where P <> 0)  Begin underscoring the text in the
               display.

<ESC>[0m or
<ESC>[m       Stop underscoring the text in the display
```

1.18 Report to the Host Command Escape Sequences (CPR, DSR)

These commands involve a query command from the host, and a response by the terminal. The query command takes the form:

```
<ESC>[Pn
```

where *P* selects the type of report requested. Two values of *P* are recognized: 5, which is a device status report, and 6, which requests a cursor position report.

The response takes the forms:

```
<ESC>[0n
```

that means "Ready, no malfunctions detected" and

```
<ESC>[Pl;PcR
```

where *Pl* is a two-digit ASCII number giving the current line (line 1 is at the top) and *Pc* is a two-digit ASCII number giving the current column.

```
Host:  <ESC>[5n       Please report status.

TE:    <ESC>[0n       Ready, no malfunctions detected.

Host:  <ESC>[6n       Please report active (cursor) position.

TE:    <ESC>[Pl;PcR   Cursor is at line Pl, column Pc.
```

1.19 VT52 Command Escape Sequences

All VT52 commands in the PS 390 Terminal Emulator, except one, take the form:

```
<ESC>c
```

The exception, Direct Cursor Addressing, is discussed in the last paragraph of this section.

```
<ESC>A  Move cursor up one position. Do not scroll.
<ESC>B  Move the cursor down one position. Do not scroll.
<ESC>C  Move cursor right one position.
<ESC>D  Move the cursor left one position.
```

`<ESC>H`	Move cursor to line 1, column 1.
`<ESC>I`	Reverse line feed; reverse scroll if at top.
`<ESC>J`	Erase to end of screen.
`<ESC>K`	Erase to end of line.
`<ESC>=`	Enter alternate keypad mode.
`<ESC>>`	Exit alternate keypad mode.
`<ESC><`	Enter ANSI mode.

Direct Cursor addressing requires a 4-character sequence. The first two characters are <ESC>Y. The next two characters indicate the line and the column to move to. The desired number is obtained by subtracting 31 (Hex 1F or Octal 37) from the ASCII character code of the character. The first character, indicating the line, will be in the range of "!" (line 1) to "8" (line 24), and the second character, indicating the column, will be in the range of "!" to "p" (column 80). For example:

```
<ESC>Y/@    Move the cursor to line 15, column 32.
```

The keypad mode commands are always recognized apart from VT52 emulation.

2. PS 390 Terminal Emulator Function Network

The actual networking of the functions that build the terminal emulator is shown in the system functions Section of *RM2 Intrinsic Functions.* This section will discuss the three main terminal emulator functions in more detail.

2.1 Keyboard Manager — (KBhandler1)

The keyboard manager takes the stream of raw bytes from the keyboard and distributes them to output queues (translating to ANSI control sequences if necessary), and toggles graphics and terminal emulator displays.

```
F:K2ANSI          Keyboard Manager - KBhandler1

Inputs:

   <1>:     Strings originating at keyboard; connected to data
            concentrator demultiplexing function.

Outputs:

    <1>: To KEYBOARD
    <2>: To SetUp
    <3>: To CHOP PARSE
    <4>: To host
    <5>: To display handler function
    <6>: Unused
    <7>: To FKEYS.
    <8>: To SPECKEYS
    <9>: To user function-networks
   <10>: Unused
   <11>: Unused

Private:

   None.
```

The first four outputs, <1> to <4>, are keyboard routes for different tasks that the keyboard performs. Output <1> ultimately goes to a user function network that has been connected to KEYBOARD. This output is used when the keyboard is in the Local (interactive) communication mode. Output <2> goes to the TE SETUP function; hitting the SETUP key or the CTRL SETUP sequence toggles this mode on and off. Output <3> is the output for the "Command" (CI) communication mode. It goes through a line editor function to chop and parse the command line for the interpretation of PS 390 commands. Output <4> is the Terminal Emulator (TE) output port and output is sent to the host computer.

The other outputs are minor and special purpose to some extent. Output <5> goes directly to the TE display data handler function for two reasons: the first is to pass commands resulting from the CLEAR/HOME key being pressed (PS 300-style keyboard only), the second is to implement the local-echo option of the terminal emulator. When outputting to this queue, the key handler expands CR (Carriage Return) to CRLF (Carriage Return/Line Feed).

Output <7> sends out the proper Qinteger when an Fkey is pressed and input to a user function network is desired via FKEYS.

Output <8> allows the cursor keys to be used in user function networks via SPECKEYS.

Output <9> allows the numeric value of the numeric keypad keys and the cursor keys to be passed to user function networks.

2.2 Terminal Emulator Display Handler (F:VT10) - ES_TE1

This function receives input from the host, from an error formatting function, and from the line editor that receives input from the keyboard in Command (CI) mode. The primary task of the data display handler is to make this input visible on the PS 390 screen.

F:VT10 **TE display handler - ES_TE1**

```
Inputs:

   <1>:Qpackets, Qmorepackets. Input to the TE.
   <2>:Qstring. Answerback string.

Outputs:

   <1>:Qpackets. Bells for the keyboard.
   <2>:Qpackets. Status, cursor reports (to host).
   <3>:Qpackets. Terminal ID (VT52 or VT100 to host)
   <4>:Qpackets. Echoed unknown escape sequences.

Private:

     None.
```

Users may send an answerback string to input <2> of ES_TE1. When the host sends ENQ (UE or %X5), the answerback string is sent to the host. As most of the input stream will have an effect on the screen, or show up as displayable data, the outputs are minor. Output <1> is used to make the expected "beep" on receipt of a ↑G (the beeper is in the keyboard).

Output <2> sends data back to the host when the function receives command sequences, such as cursor position and terminal ID (I am a VT100).

Output <3> is used to send the correct control sequence back to the host that identifies the terminal.

Output <4> is an aid for debugging and development. It sends out all command sequences that are received, but unknown by the function. Output <4> is not normally not used. When connected, it can be used to discover what kind of sequences a host program might be sending (that the terminal emulator cannot interpret) by hooking the output to a function such as Message_Display.

2.3 Terminal Emulator Setup

TE_SETUP changes the characteristics of the terminal emulator.

```
Inputs:

   <1>:   Messages from key_manager

Outputs:

   None.

Private:

    None.
```

The SETUP function gets input from the keyboard function and uses it to change the characteristics of the terminal emulator as a whole. Like the display handler function, the setup function manipulates a display structure that appears on the PS 390 screen and changes it in response to actions by the user. SETUP is interactive and uses menus and the function keys.

2.4 TE Initial Data Structures

The data structures used by the terminal emulator are set up by the CONFIG.DAT file and then completed by the function TE_BUILD.

The CONFIG.DAT file contains a color node. The color node sets the color for the characters displayed on the screen in the terminal emulator mode. The color node is accessible by sending the appropriate value to TECOLOR1.

TE_BUILD adds a set node, a 4x4 matrix, a matcon2 (to scale characters), and a set node (called the line set) to the name TD0$ that is established by the CONFIG.DAT file. From the line set, a structure is hung for each line and for the cursor. The display handler function keeps various pointers into this structure and uses them to get data on the screen, perform scrolls, etc.

3. Keyboard Communication Modes

The three modes of operation, Terminal Emulator (TE), Command (CI), and Local (KB) are all modes of operation that are established by pressing a key (or combination of keys) on the keyboard. The term "mode" is slightly misleading as it is used here. Mode is also used to describe the operation of the keypads, cursor keys, and other terminal emulator features. The modes referred to here are actually determined by what output port is used by the key_manager.

The command sequences that can be sent to <1>KBhandler1 to toggle these communication modes are:

Command	`CHAR (22) & CHAR (18)`
Local	`CHAR (22) & 'R'`
Terminal Emulator	`CHAR (22) & 'r'`

For example, to boot up in Local mode, the following command would be placed in SITE.DAT:

```
SEND CHAR (22) & 'R' to <1>KBhandler1;
```

The keys and the output that are generated from the keyboard_manager (KBhandler) in the three modes is discussed in the following section.

3.1 Keys and Outputs

In any of the modes listed below, if Keypad Keys Always or Cursor Keys Always is set, the numeric value of the keys will be sent to any user function network connected to <9>KBhandler. If Fkeys Always is set, an integer is output from FKEYS<1>.

In Local mode (KB), the numeric keypad keys will be translated into the keycap numbers if Knum is true; otherwise they will be passed out KEYBOARD as ASCII characters (or to output <9> as the numeric value of the key if Keypad Keys Always is set). Fkeys always go out the FKEYS queue as Qintegers and the cursor keys always go out to SPECKEYS as <char>xyzw (or to output <9> as <char> if Cursor Keys Always is set).

In Command mode (CI), the numeric keypad keys will be passed as numbers if Cnum is true and as sequences, (↑V<char>) if Cnum is false. Output here is only to the CI queue. Finally, Fkeys will always go out the CI queue as ↑V<char>.

The Terminal Emulator (TE) mode is the most complicated. The treatment of the numeric keypad depends on two modes: DECKPM and DECCKM. In DECKPM, when false, the keys are translated to their keycap values (numeric mode).

When DECKPM is true, the keys are translated into escape sequences. The escape sequences that are generated depend on whether VT52 is true (<ESC>?<char>) or false (<ESC>O<char>).

Fkeys are translated identical to their translation in CI mode. Otherwise, the keys become a superset of DEC's PF keys and send out <ESC>O<char> sequences like the numeric keypad in DECKPM mode.

The cursor keys send out escape sequences (unless Cursor Keys Always is set). If VT52 is true, the sequences are <ESC><char>. If the TE is emulating a VT100, then the sequence depends on DECCKM. If DECCKM is true, then <ESC>O<char> is sent, so that the cursor keys look like PF keys (or the numeric keypad in DECKPM mode); otherwise the sequence is <ESC>[<char>, which is the ANSI command sequence to move the cursor one place in the direction of the arrow.

The GRAPH and TERM keys (or CTRL GRAPH/CTRL TERM sequences on PS 390-style keyboards) allow the user to toggle the graphics and the TE displays on and off. The viewports and the set are created in the CONFIG.DAT file.

The SETUP key (CTRL SETUP sequence on PS 390-style keyboards) toggles the SETUP mode. In SETUP mode, all keys are passed to the SETUP function. When SETUP is pushed a second time, or the CTRL SETUP sequence is entered again (PS 390-style keyboards), the last use and queue are pulled.

The LINE/LOCAL key (LOCAL key on PS 390-style keyboards) is used to multiplex the keyboard between the communication modes (except SETUP). Refer to Section *IS3* for detailed descriptions of the key sequences used to change between communication modes.

The following table is an attempt to illustrate the keys, the modes, the output of the keys in the modes, and any other combinations that are useful. The representation in the table assumes that ANSI is set.

Table 10-4. *Keys, Modes, and Outputs*

	TE MODE	CI MODE	KB MODE
FUNCTION KEYS	Final 4 keys used w/ DECKPAM (EDT)	↑V char	Qinteger to FKEYS
Fkeys Always	Qinteger to FKEYS	Same as TE	Same as TE
CURSOR KEYS			
DECCKM - Set	Application functions to host	Ignored	Ignored
DECCKM - Reset (w/DECKPAM set)	Cursor control commands to host	Ignored	<char>to SPECKEYS
Cursor Keys Always	Qinteger to KBhandler<9>	Same as TE	Same as TE
Numeric KEYPAD			
DECKPNM	Numeric value passed to host	Ignored	Ignored
DECKPAM	Transmits control sequences to host for EDT utility	Ignored	Ignored
Keypad Keys Always	Qinteger to KBhandler<9>	Same as TE	Same as TE

3.2 Using the SITE.DAT File to Change Features of the Terminal Emulator

The SITE.DAT file can be used to set bootable values for the SETUP features of the terminal emulator. The following section gives the PS 390 commands that can be used to change features or defaults of the PS 390 Terminal Emulator.

TE characteristics are changed by sending sequences to <1>KBhandler1. These sequences will have the same effect as if they had been keyed in the SETUP mode of the Keyboard and Display. (Refer to Section *IS3 Operation and Communication* for a description of the SETUP feature of the Terminal Emulator.)

There are four groups of commands: Toggles, BREAK Key, Mode, and Displays, each of which is handled differently.

- Toggles

  These are TE options that have two values, true and false or on and off. In SETUP, they are changed by pressing a single Function Key that changes the present value to its opposite. To put a command in the SITE.DAT file so that the TE feature comes up in its desired value at bootup, the toggling sequence must be sandwiched between two sequences that represent the pressing of the SETUP key. The header and trailer sequence for the SETUP key is CHAR(22) & "o".

  The following chart gives the SETUP name, the definition, the default value, and the PS 390 command sequence to change the default.

Table 10-5. SETUP Toggling Sequence

Setup Name	Definition	Default Setting	Sequence to toggle
SRM	Local Echo	OFF	CHAR (22) & 'b'
Awrp	Automatic line wrap	OFF	CHAR (22) & 'c'
ANSI	ANSI sequences obeyed	ON	CHAR (22) & 'd'
VT52	VT52 mode	OFF	CHAR (22) & 'e'
KPM	Keypad Application Mode	OFF	CHAR (22) & 'f'
CKM	Cursor Key Mode	OFF	CHAR (22) & 'g'
Cnum	Keypad Numeric CI Mode	ON	CHAR (22) & 'h'
Knum	Keypad Numeric KB Mode	ON	CHAR (22) & 'i'

For example, to setup the TE for local echo (host is noecho) and for automatic line-wrap, the following command would be placed in the SITE.DAT file:

```
SEND CHAR(22) & 'o' & CHAR(22) & 'b' & CHAR (22) & 'c'
   & CHAR(22) & 'o' to <1>KBhandler1;
```

It is recommended, when possible, that the E&S private escape sequences used to set/reset the various modes of the terminal emulator be placed in the SITE.DAT file. These commands are generally more compact and take up less space on the diskette. For example, to setup the TE for local echo (host is noecho) and for automatic line-wrap, the following commands can be sent to ES_TE1:

```
Send CHAR(27) & '[>1;2h' to <1>ES_TE1;
```

- The BREAK Key

  The BREAK key, like the toggles, must be sandwiched between sequences representing the SETUP key. It also has an inner sandwich, telling SETUP that it is the BREAK key and the end of the definition. The important sequence in these two outer wrappings represents the special key designated by the user to be the BREAK key. For example, to set a key as the BREAK key, the following command would be placed in the SITE.DAT file:

```
SEND CHAR(22) & ´o´ & CHAR(22) & ´j´ & (Key sequence)
   & CHAR(22) & ´a´ & CHAR(22) & ´o´ to <1>KBhandler1;
```

  where:

  CHAR(22) & 'o' is the header/trailer sequence for the SETUP key

  CHAR(22) & 'j' is the sequence for Function Key #10 (to enter the set/BREAK key mode)

  (Key sequence) is the CHAR(22) sequence designating a user-specified key as the BREAK key

  CHAR(22) & 'a' is the sequence for Function Key #1 (exiting out of set/BREAK key)

  CHAR(22) & 'o' is the header/trailer sequence to exit SETUP.

- Mode

  To put the keyboard into Local (interactive) mode on bootup, the following should be put in the user's SITE.DAT file:

```
SEND CHAR(22) & ´R´ to <1>KBhandler1;
```

  The PS 390 normally comes up in Terminal Emulator Mode (TE) mode; that is, the keyboard outputs to the initial instance of ES_TE. To change to the other two modes (either Command or Local), the following sequences may be inserted in the SITE.DAT file. Note that these do not have to be sandwiched between SETUP key sequences.

MODE	SEQUENCE
Command	`CHAR (22) & CHAR (18)`
Local	`CHAR (22) & ´R´`
Terminal Emulator	`CHAR (22) & ´r´`

- Displays

  The two displays are the TE display and the Graphics display. They are toggled by the TERM and GRAPH keys (CRTL TERM/CTRL GRAPH on PS 390-style keyboards) and normally are on. To turn them off at boot time, special sequences may be sent.

DISPLAY	SEQUENCE
TE	`CHAR(22) & ´s´`
Graphics	`CHAR(22) & ´p´`

  For example, to turn the TE display off at boot time, the following command would be placed in SITE.DAT:

```
SEND CHAR(22) & ´s´ to <1>KBhandler1;
```

  The only TE characteristic that cannot be conveniently set by a SITE.DAT file is the size and placement of the TE display.

3.3 Using the SITE.DAT To Send Control Sequences to the Terminal

Control sequences that affect the screen display (as well as any other escape sequences) can be placed in the SITE.DAT file as ASCII sequences. The terminal emulator function ES_TE1 can accept and translate these sequences. The escape sequence in the SITE.DAT should take the following form:

```
SEND <char> n &´[P1;P2;...Pnc´ to ES_TE1;
```

where [is the control sequence introducer and **P1** through **Pn** are the parameters that may or may not be present.

4. IBM 3278 Terminal Emulation

4.1 Overview of the Environment

In the IBM 3278 interface environment, the IBM host assumes the PS 390 is an IBM 3278 display terminal attached to a 3274 Control Unit. In a normal 3274/3278 environment, application programs are able to send special characters to a 3278 terminal by packaging them in what is referred to as a Write Structured Field (WSF) envelope. E&S uses this formatting scheme to send graphical data down from the host using the Load Program Symbols option of the WSF command. This allows binary data to be sent unchanged to the PS 390. All non-WSF data are routed to the terminal emulator that performs like a 3278 display terminal.

4.2 Keyboard Communication Functions and Modes

The three keyboard modes, Terminal Emulator (TE), Command (CI), and Local (KB) are all modes of operation that are established by pressing a key (or combination of keys) on the keyboard. The Terminal Emulator mode allows use of the PS 390 as an IBM terminal. While in the TE mode, the screen is formatted as an IBM 3278 terminal. The Command mode permits the PS 390 to be used as an independent processor. In the command mode, the screen is formatted as a DEC VT100 terminal. Local mode allows the keyboard to be used as a peripheral graphics device. In Local mode the function keys and standard keyboard keys may act as inputs to any user-created function networks that are connected to them.

The modes referred to here are actually determined by what output port is used by the function F:IBM_KEYBOARD, called the 3278 terminal emulator keyboard handler.

The keyboard handler is a submodule of the IBM 3278 terminal emulator. This function receives bytes of character data from the keyboard, distributes them to the output queues, and translates them to IBM scan codes or to ASCII characters if necessary. Translations are performed to support the keyboard used (either VT100 style or IBM) and the output port and destination the data will be sent to. It also toggles the graphics and terminal emulator displays.

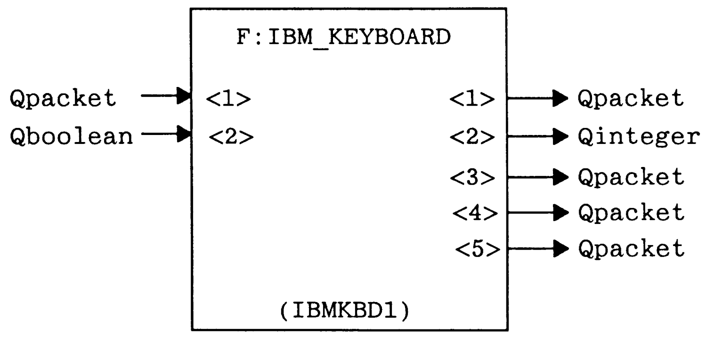


Figure 10-1. F:IBM_KEYBOARD

F:IBM_KEYBOARD accepts character packets from the keyboard on input <1> and based on the mode (either Terminal Emulator, Command, or Local), outputs packets for use by the function network, the line editor, or an IBM host. Packets of characters for the KEYBOARD function are output on <1>. Qintegers to be sent to the FKEYS function are output on <2>. Qpackets of characters to be sent to the function, SPECKEYS, are output on <3>. Qpackets of characters for the line editor are output on <4>. Qpackets of IBM scan codes for an IBM host are output on <5>.

Input <2> accepts a Boolean that indicates which type of keyboard is being used.

```
True = IBM-style keyboard
False = VT100-style keyboard
```

At system configuration, a VT100-style keyboard is specified; so, if an IBM-style keyboard is being used, the following PS 390 command should be entered in the SITE.DAT file:

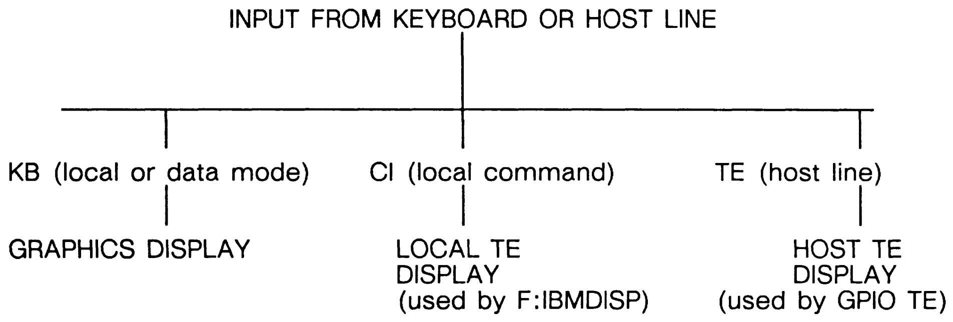
```
SEND TRUE TO <2>IBMKBD1;
```

4.3 Data Structures

The three main output ports of the keyboard handler all affect a different data display structure. The data structures used by the terminal emulator are set up by the CONFIG.DAT file and then completed by the function TE_BUILD.

The CONFIG.DAT file contains a color node. The color node determines the color of the characters on the screen in the terminal emulator mode. The color node is accessible by sending the appropriate value to TECOLOR1.

A simplified diagram of the display structure created by the terminal emulator is shown below:


INPUT FROM KEYBOARD OR HOST LINE
KB (local or data mode)
CI (local command)
TE (host line)
GRAPHICS DISPLAY
LOCAL TE DISPLAY (used by F:IBMDISP)
HOST TE DISPLAY (used by GPIO TE)

The GPIO (the I/O processor used for communication with the IBM host) is able to differentiate between data that is bound for the Host Screen Buffer (3278 terminal emulation) and data that is bound for the PS 390 command interpreter (graphical data). All data bound for the CI is packaged in WSF envelopes. (Refer to Section *RM5 Host Communications* for information on WSF commands and data flow from the host system.) Upon receiving information from the host, the GPIO differentiates graphical data from TE data by the WSF command; anything not in a WSF command is TE data and goes directly to the (Host) Screen Buffer.

The local TE display is set up by the F:IBMDISP function.

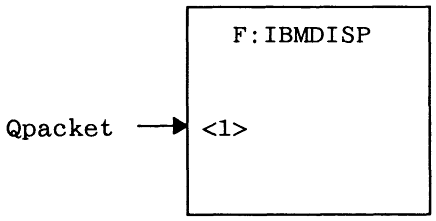


Figure 10-2. F:IBMDISP

F:IBMDISP accepts packets of ASCII characters on input <1>. Then, it either inserts their equivalent IBM screen code into the local screen buffer used by the Command mode of the terminal emulator or causes the cursor position to be adjusted in the case of a carriage return, a line feed, or a back space.

4.4 Indicator Characters

The PS 390 supports indicator characters that indicate the status (active mode, software exception, etc.) of the PS 390. These characters appear on the right side of the indicator line, and are defined as follows:

H Indicates that the keyboard is communicating with the host.

C Indicates that the keyboard is communicating with the CI.

L Indicates that the keyboard is communicating with user function networks.

S Indicates that the keyboard is in the SETUP mode, i.e., the SETUP key has been pressed.

G Indicates that the graphics display is active.

¢ Indicates that the GPIO was unable to establish communications with the host.

G̸ Indicates that the GPIO timed out.

† Indicates that the CAPS LOCK feature is active.

These indicator characters (with the exception of the SETUP and the two error indicators) may be removed from the screen by using the SETUP mode of the keyboard.

4.5 Setup Mode for the Terminal Emulator

The SETUP mode for the 3278 TE is accessed by pressing CTRL SETUP. In SETUP, the Function Keys on the keyboard are used to toggle or adjust screen display features. CTRL SETUP must be pressed again, after the appropriate adjustments are made, to exit the SETUP mode.

SETUP can be entered in any communication mode and can be used to make the following adjustments:

FKey #1 Pressing this key increases the intensity of the screen.

FKey #2 Pressing this key decreases the intensity of the screen.

FKey #3 Pressing this key raises the contrast of the screen.

FKey #4	Pressing this key lowers the contrast of the screen.
FKey #5	Pressing this key toggles in and out of the CAPS LOCK mode. While in CAPS LOCK, all standard keypad keys output their shifted value. (This is for IBM-style keyboards only.)
FKey #6	Pressing this key toggles the display of PS 390 characters. The default is the display of characters.
Fkey #7	Toggles the display of the host indicator characters. The default is the display of characters.
Fkey #8	Toggles the display of the cursor. Default is display of the cursor.

Function keys F9 and F10 are used in conjunction with the PS 390/IBM 3250 Interface. Information on the use of these keys is available in the PS 300/IBM 3250 Interface User's Manual.

4.6 Using the SITE.DAT File to Change Features of the Terminal Emulator

The adjustments made in SETUP can be entered as PS 390 commands in the SITE.DAT file to set the appropriate characteristics at boot time.

The list below shows the characters that should be entered into the SITE.DAT file for each feature.

For VT100 style keyboards, the appropriate character(s) must be inserted between a ´↑Vo ↑Vo´ header and trailer sequence. (↑Vo is a CTRL V lowercase "o" sequence.)

FEATURE	CHARACTERS TO BE ENTERED INTO SITE.DAT
Raise Intensity	SEND ´↑Vo↑Va↑Vo ´ TO <1>IBMKBD1;
Lower Intensity	SEND ´↑Vo↑Vb↑Vo ´ TO <1>IBMKBD1;
Raise Contrast	SEND ´↑Vo↑Vc↑Vo ´ TO <1>IBMKBD1;
Lower Contrast	SEND ´↑Vo↑Vd↑Vo ´ TO <1>IBMKBD1;
Set/Reset Caps Lock	SEND ´↑Vo↑Ve↑Vo ´ TO <1>IBMKBD1;
Set/Reset Local Indicators	SEND ´↑Vo↑Vf↑Vo ´ TO <1>IBMKBD1;
Set/Reset Host Indicators	SEND ´↑Vo↑Vg↑Vo ´ TO <1>IBMKBD1;
Set/Reset Cursor	SEND ´↑Vo↑Vh↑Vo ´ TO <1>IBMKBD1;
Set 3250 Mode	SEND ´↑Vo↑Vi↑Vo ´ TO <1>IBMKBD1;
Set PS 390 Mode	SEND ´↑Vo↑Vj↑ Vo´ TO <1>IBMKBD1;

For IBM-style keyboards, the appropriate characters must be inserted between a CHAR(130)&CHAR(n)&CHAR(130) sequence, where &CHAR(n) is the character sequence(s) for the feature.

FEATURE	CHARACTERS TO BE ENTERED INTO SITE.DAT
Raise Intensity	SEND CHAR(130)&CHAR(145)&CHAR(130) TO <1>IBMKBD1;
Lower Intensity	SEND CHAR(130)&CHAR(146)&CHAR(130) TO <1>IBMKBD1;
Raise Contrast	SEND CHAR(130)&CHAR(147)&CHAR(130) TO <1>IBMKBD1;
Lower Contrast	SEND CHAR(130)&CHAR(148)&CHAR(130) TO <1>IBMKBD1;
Set/Reset Caps Lock	SEND CHAR(130)&CHAR(149)&CHAR(130) TO <1>IBMKBD1;
Set/Reset Local Indicators	SEND CHAR(130)&CHAR(150)&CHAR(130) TO <1>IBMKBD1;
Set/Reset Host Indicators	SEND CHAR(130)&CHAR(151)&CHAR(130) TO <1>IBMKBD1;
Set/Reset Cursor	SEND CHAR(130)&CHAR(152)&CHAR(130) TO <1>IBMKBD1;
Set 3250 Mode	SEND CHAR(130)&CHAR(153)&CHAR(130) TO <1>IBMKBD1;
Set PS 390 Mode	SEND CHAR(130)&CHAR(154)&CHAR(130) TO <1>IBMKBD1;

When inserting multiple SETUP options in the SITE.DAT file, as many values as needed should be entered in between the header and trailer sequences.

For example, on VT100 style keyboards

```
SEND '↑Vo↑Va↑Vc↑Vf↑Vo' TO <1>IBMKBD1;
```

would raise screen intensity, raise screen contrast, and toggle the local indicator display.

For IBM style keyboards, this sequence would be

```
SEND CHAR(130)&CHAR(145)&CHAR(147)&CHAR(150)&CHAR(130)
TO <1>IBMKBD1;
```

The horizontal line running across the bottom of the terminal display can be removed by entering the following PS 390 command in the SITE.DAT file:

```
IBMLINE$:=NIL;
```

Another feature that can be changed in the SITE.DAT file is status of the displays affected by CTRL GRAPH and CTRL TERM sequences.

VT100 style Sequences	DESCRIPTION
`SEND ↑Vp TO <1>IBMKBD1;`	Toggles TERM display
`SEND ↑Vs TO <1>IBMKBD1;`	Toggles GRAPH display

IBM style Sequences	DESCRIPTION
`SEND CHAR(83) TO <1>IBMKBD1;`	Toggles TERM display
`SEND CHAR(82) TO <1>IBMKBD1;`	Toggles GRAPH display

RM11. SYSTEM ERRORS

Section RM11 System Errors

This section provides a description of the system error messages that you may encounter during standard operation of the PS 390 graphics system. Errors may be written to the debug terminal, to the keyboard LEDs, or to the Crash Dump file. There are three types of error messages, listed in the following three tables.

NOTE

The tables list the error messages for PS 390 systems using either DEC or IBM host computers. It is noted in the tables where the message is host-specific.

The first table lists the error number and brief description of the traps or software induced exceptions that might cause the system to fail.

The second table lists the error numbers (with error definitions) of system errors that might be caused when you use the user-written function (UWF) facility.

The third table is a comprehensive list of the system error numbers. Most system errors are generated only during the development process of the graphics firmware and are rarely seen during normal system operation.

NOTE

Notify E&S Customer Engineering Software Support when any error numbers are reported that are E&S firmware errors (shown in Table 11-1 or Table 11-3).

Table 11-1. PS 390 Traps and Definitions

NUMBER	DEFINITION
0	Not enough available memory to come up or handle request.
1	E&S firmware error.
2	Memory corrupted or over-written (could be caused by UWF).
3	Memory corrupted or over-written (could be caused by UWF). Message for systems using IBM host only.
5	Attempt to wait on queue when function is waiting on another device (CLOCK, I/O)(could be caused by UWF).
6	System errors (refer to Table 11-3).
7	Double-bit mass memory error if address on LEDs is between 200 and 300; unexpected interrupt on a vector with no routine if address is between 300 and 400.
8	Usually indicates double-bit mass memory error. If address on LEDs is 22C, error occurred on memory card 200000-300000. If address is 23C, error occurred on memory card 300000-400000 and so forth. Message for systems using DEC host only.
9	E&S Firmware Error
10	Memory corrupted or over-written (could be caused by UWF).
11	E&S firmware error.
12	Pascal in-line runtime error: usually caused by Case statement in Pascal with no Otherwise clause (could be caused by UWF).

Table 11-2. User-Written Function Error Descriptions

ERROR NUMBER	DEFINITION
SYSTEMERROR #7F	Exited function before re_queuing function (not following template).
SYSTEMERROR #80	Bad parameter passed to text utility routine: Text_text, B1 < 0.
SYSTEMERROR #81	Bad parameter passes to text utility routine: Char_text, b < 0.
SYSTEMERROR #81	Bad parameter passes to text utility routine: Char_text, b < 0. Message for systems using IBM host only.
SYSTEMERROR #8E	Bad parameters passed to Updates utilities: AnnounceUpdate List tail = nil; head <> nil.
SYSTEMERROR #B9	Nil or invalid parameter passed to Illegal_Input handling routines.
SYSTEMERROR #C9	User written function stack overflow.
SYSTEMERROR #CB	Improper redefinition of user written function name.
SYSTEMERROR #D9	Call to Ckinputs has Nmin < 0.
SYSTEMERROR #DA	Call to Ckinputs has Nmin > Nmax.
SYSTEMERROR #DB	Call to Ckinputs has Nmax > total number of inputs for function.
SYSTEMERROR #DE	Multiple call to Qsendcopymsg on the same input.
SYSTEMERROR #E0	Function was not in state Running when Ckinputs was called; Cleaninputs returned a FALSE and still called Ckinputs; Cleaninputs was not called before calling Ckinputs the second time.

Table 11-2. User-Written Function Error Descriptions (continued)

ERROR NUMBER	DEFINITION
SYSTEMERROR #E1	Function was not in state Mid_running when Cleaninputs was called.
SYSTEMERROR #E9	Qillmessage, or Qillvalue was called for input which does not exist.
SYSTEMERROR #EA	Qillmessage, or Qillvalue was called for input which was already dealt with; previous call to Qillmessage, Qillvalue, or Qsendcopymsg.
SYSTEMERROR #110	Tolerance on FCnearzero is too small.
SYSTEMERROR #111	Set node has no dummy control block.
SYSTEMERROR #68	Possible overwrite of block boundary: Sending to an unrecognized Namedentity.
SYSTEMERROR #A1	Possible overwrite of block boundary: AppendVector, Invalid Acpdata type.
SYSTEMERROR #92	Possible overwrite of block boundary: Byte Index Invalid Acpdata type.
SYSTEMERROR #9C	Possible overwrite of block boundary: Unrecognized type of Namedentity.
SYSTEMERROR #9D	Possible overwrite of block boundary: Hasstructure.
SYSTEMERROR #A3	Possible overwrite of block boundary: Nomemsched, Bad.Status for a fcn.
SYSTEMERROR #103	Possible overwrite of block boundary: Curfcn was not active at entry.
SYSTEMERROR #109	Possible overwrite of block boundary: ContBlock, nil block.

Table 11-2. User-Written Function Error Descriptions (continued)

ERROR NUMBER	DEFINITION
SYSTEMERROR #10D	Possible overwrite of block boundary: GetVector, Not an Acpdata block.
SYSTEMERROR #10E	Possible overwrite of block boundary: GetVector, Not a vector Acpdata block.

Motorola Exceptions:

These exceptions could be due to E&S software exceptions or to UWF memory overwrites.

Exception 2	Bus Error.
Exception 3	Address Error.
Exception 4	Illegal Structure.

The following table is the comprehensive list of system error messages. The messages are listed numerically, but can show one of two error types:

- Possible UWF Error — software exceptions possibly caused by use of the UWF facility. If UWF is not used, notify Customer Engineering Software Support when you get this message.

- E&S Firmware Error — software exceptions that indicate that Customer Engineering Software Support should be notified.

Table 11-3. List of System Error Messages

ERROR NUMBER	DEFINITION
#64	E&S Firmware Error
#65	E&S Firmware Error
#66	E&S Firmware Error
#67	E&S Firmware Error
#68	Possible UWF Error
#69	E&S Firmware Error
#6A	E&S Firmware Error
#6B	E&S Firmware Error
#6C	E&S Firmware Error
#6	E&S Firmware Error
#6E	E&S Firmware Error
#6F	E&S Firmware Error
#70	E&S Firmware Error
#71	E&S Firmware Error
#72	E&S Firmware Error
#73	E&S Firmware Error
#74	E&S Firmware Error
#75	E&S Firmware Error
#76	E&S Firmware Error
#77	E&S Firmware Error
#78	E&S Firmware Error
#79	E&S Firmware Error
#7A	E&S Firmware Error

Table 11-3. List of System Error Messages (continued)

ERROR NUMBER	DEFINITION
#7B	E&S Firmware Error
#7C	E&S Firmware Error
#7D	E&S Firmware Error
#7E	E&S Firmware Error
#7F	Possible UWF Error
#80	Possible UWF Error
#81	Possible UWF Error
#85	E&S Firmware Error
#86	E&S Firmware Error
#87	E&S Firmware Error
#88	E&S Firmware Error
#8A	E&S Firmware Error
#8D	E&S Firmware Error
#8E	Possible UWF Error
#8F	E&S Firmware Error
#90	E&S Firmware Error
#91	E&S Firmware Error
#92	Possible UWF Error
#93	E&S Firmware Error
#94	E&S Firmware Error
#95	E&S Firmware Error
#96	E&S Firmware Error
#97	E&S Firmware Error

Table 11-3. List of System Error Messages (continued)

ERROR NUMBER	DEFINITION
#98	E&S Firmware Error
#99	E&S Firmware Error
#9C	Possible UWF Error
#9D	Possible UWF Error
#9E	E&S Firmware Error
#A1	Possible UWF Error
#A3	Possible UWF Error
#A9	E&S Firmware Error
#AA	E&S Firmware Error
#AB	E&S Firmware Error
#AC	E&S Firmware Error
#AD	E&S Firmware Error
#AE	E&S Firmware Error
#AF	E&S Firmware Error
#B0	E&S Firmware Error
#B3	E&S Firmware Error
#B4	E&S Firmware Error
#B8	E&S Firmware Error
#B9	Possible UWF Error
#BA	E&S Firmware Error
#BD	E&S Firmware Error
#BF	E&S Firmware Error
#C0	E&S Firmware Error

Table 11-3. List of System Error Messages (continued)

ERROR NUMBER	DEFINITION
#C1	E&S Firmware Error
#C2	E&S Firmware Error
#C3	E&S Firmware Error
#C9	Possible UWF Error
#CA	E&S Firmware Error
#CB	Possible UWF Error
#CC	E&S Firmware Error
#CD	E&S Firmware Error
#CF	E&S Firmware Error
#D0	E&S Firmware Error
#D1	E&S Firmware Error
#D2	E&S Firmware Error
#D3	E&S Firmware Error
#D4	E&S Firmware Error
#D5	E&S Firmware Error
#D6	E&S Firmware Error
#D7	E&S Firmware Error
#D8	E&S Firmware Error
#D9	Possible UWF Error
#DA	Possible UWF Error
#DB	Possible UWF Error
#DC	E&S Firmware Error
#DE	Possible UWF Error

Table 11-3. List of System Error Messages (continued)

ERROR NUMBER	DEFINITION
#DF	E&S Firmware Error
#E0	Possible UWF Error
#E1	Possible UWF Error
#E2	E&S Firmware Error
#E5	E&S Firmware Error
#E6	E&S Firmware Error
#E7	E&S Firmware Error
#E8	E&S Firmware Error
#E9	Possible UWF Error
#EA	Possible UWF Error
#EB	E&S Firmware Error
#ED	E&S Firmware Error
#EE	E&S Firmware Error
#EF	E&S Firmware Error
#F0	E&S Firmware Error
#F1	E&S Firmware Error
#F3	E&S Firmware Error
#F6	E&S Firmware Error
#F7	E&S Firmware Error
#F8	E&S Firmware Error
#F9	E&S Firmware Error
#FC	E&S Firmware Error
#FD	E&S Firmware Error

Table 11-3. List of System Error Messages (continued)

ERROR NUMBER	DEFINITION
#FE	E&S Firmware Error
#FF	E&S Firmware Error
#100	E&S Firmware Error
#101	E&S Firmware Error
#102	E&S Firmware Error
#103	Possible UWF Error
#104	E&S Firmware Error
#105	E&S Firmware Error
#106	E&S Firmware Error
#107	E&S Firmware Error
#108	E&S Firmware Error
#109	Possible UWF Error
#10A	E&S Firmware Error
#10B	E&S Firmware Error
#10C	E&S Firmware Error
#10D	Possible UWF Error
#10E	Possible UWF Error
#10F	E&S Firmware Error
#110	Possible UWF Error
#111	Possible UWF Error
#112	E&S Firmware Error
#113	E&S Firmware Error
#114	E&S Firmware Error

Table 11-3. List of System Error Messages (continued)

ERROR NUMBER	DEFINITION
#115	E&S Firmware Error
#116	E&S Firmware Error
#117	E&S Firmware Error
#118	E&S Firmware Error
#119	E&S Firmware Error
#120	E&S Firmware Error
#121	E&S Firmware Error
#122	E&S Firmware Error
#123	E&S Firmware Error
#124	E&S Firmware Error
#125	E&S Firmware Error
#126	E&S Firmware Error

RM12. DIAGNOSTIC UTILITIES

CONTENTS

Section RM12
Diagnostic Utilities

Diagnostic utilities and commands are used to back up diskettes and for diskette file management. This section explains accessing the utility program that contains the commands and then lists and gives a short description of the commands. It also provides the steps used to back up the graphics firmware diskettes or any other system diskettes.

1. Diagnostic Utility Commands

The utility program in the diagnostic operating system contains commands used to format diskettes, and to check, copy, delete, modify, download, and send back files. The utility program also has terminal emulator capabilities. This section explains loading the diagnostic utility diskette in order to access the commands, then lists the utility commands available.

1.1 Loading the Diagnostic Utility Diskette

To access the utility commands load the diagnostic utility diskette using the following steps:

1. Power-off the system, ensuring the activity light is off.
2. Dismount any diskettes in the PS 390 disk drives.
3. Insert the diagnostic utility diskette into drive 1. Ensure the E&S label on the diskette faces right and the covered write-protect slot faces up.
4. Power-up the PS 390 control unit and display. A VT100-compatible auxiliary terminal or the PS 300 keyboard with LEDs may be used to enter in the diagnostic commands. The auxiliary terminal is the preferred equipment to use since it fully displays system prompts.

NOTE

If an auxiliary terminal is used and you are operating it at 300 baud, connect it to one of the available ports on the control unit connector panel. Usually, Port 3 (debug port) is used.

5. Wait until the PS 390 finishes its power-on confidence tests and the auxiliary terminal displays "O" and beeps before continuing.
6. Hold down the CTRL key while repeatedly typing P:

   ```
   <CTRL>P
   ```

   until the system responds with:

   ```
   PS 300 Diagnostic operating system Ax.Vxx
   Disk name = PS 300 Diagnostic Disk X Ax.Vxx
   Type "HELP" for help.
   =
   ```

   The = prompt indicates the diagnostic operating system has been successfully loaded.
7. Select the utilities program by typing:

   ```
   UTILITY <CR>
   ```

1.2 Selecting Utility Commands

The following message is displayed when the utility program has been successfully loaded.

```
=Utility
UTILITY; 1 loaded
PS 300 file and download Utility Px.Vxx
Type HELP for additional help.
Utility>
```

Enter a command at the Utility> prompt. The command is an alphabetic string that is long enough to identify the command. For example, you can type in the full word CHECK, or abbreviate it to CH, to call the CHECK command. If the first character in the entered command is not alphabetic, or if the first word in the entered command is incorrect, the system responds with:

```
Invalid command.
```

The system prompts you for any required parameters that are not entered by the operator. Those commands containing parameters require more than one line when they are entered.

When you enter the command, the utility program steps you through a series of prompts that completes the command.

1.3 Utility Commands

The following file utility commands are available:

CHECK	Reads the entire diskette to check for diskette errors and to determine if the file structure is valid.
COMPARE	Compares two diskettes or two files to determine if they are the same.
COMPRESS	Compresses a diskette by copying each file over any empty space on the diskette until all empty space resides in one contiguous block at the end of the diskette.
COPY	Copies a file from one diskette to another diskette or copies a file from one place on a diskette to another place on the same diskette.
COPYDISK	Copies the contents of an entire diskette onto another diskette.
CREATE	Creates a file from data in memory.
CREATEBOOT	Creates a boot file from an existing file.
DATE	Displays and/or changes the date.
DELETE	Deletes a file.
DIRECTORY	Displays the diskette directory.
DRIVE	Selects a diskette drive.
DUMP	Dumps a file from the diskette into memory.
EXIT	Returns to the Diagnostic Operating System monitor.
FORMAT	Formats and initializes a diskette.

FREE	Indicates the number of free blocks on a diskette.
HELP	Displays a list of available commands and information about each command.
INITIALIZE	Initializes a diskette without formatting it.
MEMORY	Displays memory size and allows use of either local or mass memory.
MODIFY	Modifies the host communication parameter values for baud rate, parity, port number, etc.
PURGE	Deletes all but the latest version of each file or all but the latest version of one specific file from the diskette.
REMOVE	This is a query "delete". It will selectively delete any files on the disk as it prompts the user through the file names.
RENAME	Renames a file.
RENAMEDISK	Changes the diskette title.
RESTORE	Restores one of the saved (see SAVE command) files containing the host communication value parameters.
SAVE	Saves host communication parameter values modified using the MODIFY command.
SENDBACK	Transfers a file from the PS 390 to the host.
TERMINAL	Software resident on the diagnostic diskette lets the user access a line to the host system. The communication parameters must be correct for this command to work.
TRANSFER	Transfers a file from the host to the PS 390.
TYPE	Types the contents of a file to the terminal.

Use the HELP command for a brief description of the function and syntax of each Utility command.

2. Backing Up Firmware and Diagnostic Diskettes

Backup copies should be made of the graphics firmware or any of the PS 390 diskettes. The diskettes should be copied as soon as they are received. A blank diskette(s) must be available to use as the copy disk(s). A PS 300 keyboard with LEDs, or an auxiliary terminal may be used when backing up. The auxiliary terminal is the preferred equipment to use since it fully displays system prompts.

You perform the following steps when you do a backup:

- Load the PS 390 diagnostic utility diskette.
- Access the utility program on the diagnostic diskette.
- Format a blank diskette to be used as the copy diskette.
- Store data from the original diskette for transfer to the copy diskette using either mass memory or local memory.

2.1 Formatting the Destination Diskette

The utility program in the diagnostic operating system is used to format the blank (destination) diskette and copy the PS 390 graphics firmware.

Format the blank diskette as follows:

1. Load the diagnostic diskette and access the utility program following the procedure described in sections 1.1 and 1.2.
2. Dismount the diagnostic software diskette.
3. Mount the blank diskette in the diskette drive. Ensure the write-protect tape has been removed from the diskette.
4. Type:

   ```
   FORMAT
   ```

   The system responds with:

   ```
   Utility> Format
   ENTER DISK NAME
   ```

5. Although the system asks for a disk name, a response is not necessary. The firmware or diagnostic diskette is copied onto the destination diskette, name included. Press RETURN to continue.

6. The system then formats the diskette. If the diskette is write-protected, the system returns with this message:

   ```
   *****ERROR:  Disk write protected.
   ```

   If this message appears, make sure the write-protect tape has been removed. If the tape has been removed, and the message still appears, use a new diskette.

7. When the destination diskette is successfully formatted, the Utility> prompt is displayed.

Disk formatting difficulties are usually the result of a bad disk or faulty diskette mounting in the drive. Use the CHECK command to determine if a diskette has been properly formatted.

2.2 Copying the PS 390 Diskettes

You can use either mass memory or local memory to copy the graphics firmware or diagnostic diskettes. It is faster to copy diskettes using mass memory and both disk drives. The system prompts you during the copy at each step.

2.2.1 Copying Using Mass Memory

Initialize mass memory by typing:

```
MEM
```

The system responds with:

```
Memory is currently set to use xxxK of local memory.
Do you want to change using mass memory?
```

You must respond with YES or Y to use mass memory. NO is the default answer. If you respond with a RETURN, NO, or N, the system uses local memory to copy.

When you respond with YES or Y, the system displays the current amount of mass memory available:

```
xxxK of memory is available in mass memory.
```

and initializes mass memory to be used in the COPYDISK command. Perform the following steps to complete the copy.

1. Type:

   ```
   COPYDISK
   ```

2. The system responds with:

   ```
   Copy using 1 or 2 drives?
   xxxK of memory is available in mass memory.
   ```

3. Type:

   ```
   2
   ```

4. The system prompts:

   ```
   Enter source drive number.
   ```

   Enter the number of the drive containing the source diskette. The diskette drives are numbered 1 and 2. The system then prompts:

   ```
   Enter destination drive number:
   ```

   Enter the number of the drive containing the newly formatted diskette.

5. The system prompts:

   ```
   Please insert source and destination disks, then press RETURN.
   ```

6. The system loads the data from the source file into memory and copies it onto the destination diskette. When the copy is complete, the system prompts:

   ```
   The disk has been copied.
   Do you want another copy of the same disk?
   ```

Repeat the procedure as needed.

2.2.2 Copying Using JCP Local Memory

If mass memory is not available for temporary use with the COPYDISK utility, the system uses the JCP local memory for temporary storage. The system uses the same prompts that appear during the mass memory copy procedure.

To use JCP local memory when copying, enter RETURN, NO, or N at the system prompt:

```
Do you want to change using mass memory?
```

Then follow the system prompts to complete the copy procedure.

2.3 Error Messages During Copying

There are several error messages that may appear during COPYDISK. If the system displays the following,

```
*****ERROR:  Record not found during write.
```

reformat the diskette and try COPYDISK again.

Refer to the list of Utility commands at the front of this section for more commands that may be helpful in backing up the PS 390 diskettes.

2.4 Checking the Copy

Use the CHECK command to determine if a diskette has been properly formatted. The CHECK command responds with a detailed report of the number of blocks in the header, footer, and body of each file. The message appears as:

```
Header              0
Directory           1
File Name.Ex_;26    2-43
* Empty *           44-719
```

When the system displays the Utility> prompt, and no error messages appear, the diskette has been properly formatted.

When file copying is complete, use the utility DIRECTORY to check if all files were copied from the source disk. The CHECK utility can be used to read the diskette and display the name and number of sectors of each file, or the COMPARE utility can be used to compare the newly copied disk with the source disk.

2.5 The DELETE Command

Use the DELETE command to delete a file from the diskette. This utility command should be used to delete the original SITE.DAT file from the copy of the graphics firmware before downloading the new version. If the extension is not specified, the first file found on the diskette that has a matching file name and version number is deleted. The version number must always be specified.

The following is an example deleting the SITE.DAT;4 file from the diskette:

```
Utility>DELETE
Enter name of file to be deleted:  SITE.DAT;4
File deleted successfully.
Utility>
```

The DELETE command may also be entered on one line:

```
Utility>DELETE SITE.DAT;4
```

The file name must be valid, and must include a version number. If the version number is not specified, the system advises the operator of the error with the following message:

```
Error, version number must be specified.
```

When a file is deleted it no longer exists on the diskette, and that space becomes available for other files.

NOTE

For information on using the utility commands to download a file from the host, refer to the example in TT2 Helpful Hints, *How to copy Files Between the Host and the PS 390.*

RM13A. INTERACTIVE DEVICES

PS 300 STYLE

CONTENTS

ILLUSTRATIONS

TABLES

Section RM13A
Interactive Devices
PS 300 Style

Two sets of interactive devices are available with the PS 390: the PS 300-style devices and the PS 390-style devices. Interactive devices from the two styles cannot be mixed with the exception of the data tablets and the optical mouse, which are common to both styles.

The PS 300-style interactive devices include:

- Keyboard with LEDs
- Control dials unit with LEDs
- Function buttons unit
- Data tablet (6 by 6 or 12 by 12) with puck
- Optical mouse

The light pen is not supported on the PS 390.

The PS 390 interfaces with the interactive devices through the peripheral multiplexer which supplies the power to the interactive devices and serves as their input/output path to the PS 390. The peripheral multiplexer combines the signals from the interactive devices and transmits them to the PS 390.

This section describes the PS 300-style interactive devices and peripheral multiplexer. Section *RM13B* describes the PS 390-style interactive devices and peripheral multiplexer.

1. The Peripheral Multiplexer

The peripheral multiplexer serves as the connection point between the PS 390 system and the interactive devices. It provides power to the interactive devices and combines their signals and transmits them to the PS 390. It also routes any signals which the the system may send back to the appropriate interactive device.

The peripheral multiplexer is housed in a metal box which fits beneath the raster display pedestal. The interactive devices connect to the five connectors on the front of the multiplexer. Each connector is uniquely dedicated to a specific interactive device.

Figure 13A-1 shows the peripheral connections for the PS 300-style peripheral set. Figure 13A-2 shows the backside connectors and plugs for the peripheral multiplexer.

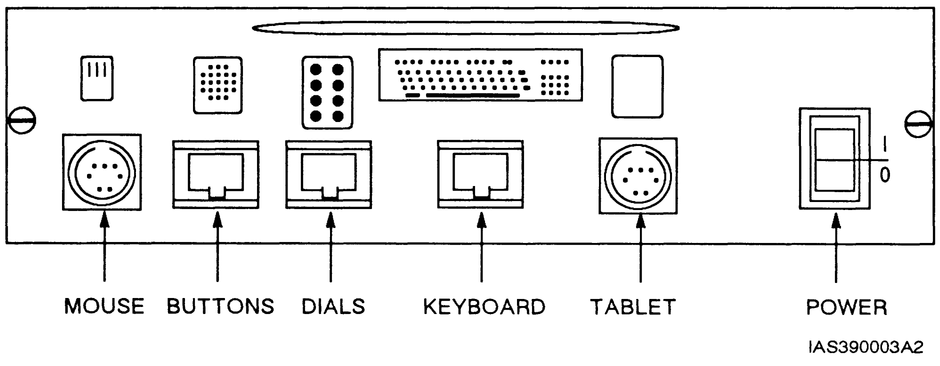


Figure 13A-1. Front Panel of Peripheral Multiplexer

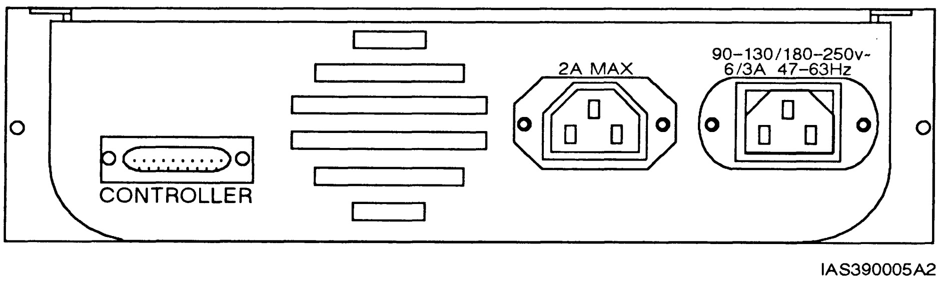


Figure 13A-2. Back Panel of Peripheral Multiplexer

1.1 Functional Characteristics

The peripheral multiplexer consists of a circuit card which is connected to five input ports and one output port. The five input ports support the following interactive devices:

- Keyboard with LEDs
- Control dials unit with LEDs
- Function buttons unit
- Data tablet (6 by 6 or 12 by 12) with cursor
- Optical mouse

The peripheral multiplexer receives input data from the interactive devices and multiplexes the data through an RS-232C output port to the PS 390. It also accepts the multiplexed data from the terminal controller, demultiplexes the data, and routes the data to the appropriate interactive devices.

1.2 Data Framing and Transmission Rates

The data sent to and from the peripheral multiplexer is asynchronous data with each byte containing eight data bits with no parity, one start bit and one stop bit. The data transmission rate of the peripheral multiplexer to and from the PS 390 is 19,200 baud. The transmission rates between the interactive devices and the peripheral multiplexer are shown in Table 13A-1.

Table 13A-1. Interactive Device Transmission Rates

Device	Baud Rate
Keyboard Port x'B1'	2400 Baud
Control Dials Port x'B2'	9600 Baud
32 Func. Buttons Port x'B3'	9600 Baud
Mouse Port x'B4'	9600 Baud
Data Tablet Port x'B6'	9600 Baud

2. Keyboard

The main function of the keyboard is the generation and transmission of ASCII displayable characters, ASCII control characters, and PS 390 system sequences. This data is transmitted to the JCP, the controlling system processor that is located in the PS 390 control unit. The transmitted data may ultimately specify displayed characters, commands, menu/table selections, etc.

The keyboard also displays full-line or segmented alphanumeric messages on a 1 to 96-character LED array. These displayed characters most often function as labels for the keyboard's 12 user-programmable function keys. The LED characters may also be used "in tandem" to present a single message up to 96 characters long.

2.1 Physical Configuration

The keyboard is a modular unit that connects to the system through a single interface cable. Like the other interactive devices, the keyboard is microprocessor-controlled to provide limited local processing capabilities.The processor in the keyboard controls LED displays and I/O data transmissions.

The keyboard unit contains a keyboard, an LED display, and a keyboard interface. The assembled keyboard measures 21.1 inches (53.6 cm) long by 8.25 inches (20.9 cm) deep. The keyboard stands 3.5 inches (8.9 cm) high on four rubber feet. The system's audible alarm sounds through a speaker.

The LEDs are configured in a single row above the twelve keyboard function keys. They are arranged in twelve 8-character groups. Each LED group may serve as a label for its associated function key, or all LED characters may be used together to display a single message. A space of one character separates each 8-character LED group from the next.

An 8-conductor, flexible cable with locking modular plugs connects the keyboard to the peripheral multiplexer. The cable is similar in function and appearance to a standard telephone "flex" cord. The cable may be stretched to permit many different work station arrangements. The modular plugs are identical, allowing the cable to be connected in either direction.

The keyboard should be grounded, and provision for this has been made on the peripheral multiplexer.

2.2 Data Entry

The 95 keys fall into eight general categories.

- Keyboard function control
- Alphabetic
- Standard numeric
- Special character
- Terminal function
- Function
- Numeric/application mode
- Device control

Note

When instructions are given to press two or more keys simultaneously, the key sequence will be shown in italics. For example, *CTRL V* means that the CTRL and V keys are pressed simultaneously.

2.2.1 Keyboard Function Control Keys

The keyboard function control keys are unencoded, local controls, and include the SHIFT and CTRL keys. No codes are transmitted when these keys are pressed individually or in combination with each other. The keyboard function control keys are used to modify the codes transmitted by other keys. When either SHIFT key is pressed simultaneously with a displayable character key, the uppercase code for that key is generated. If the key does not have an uppercase function, the SHIFT key is ignored. For example, pressing the A key causes the binary code B´01100001´ for the character **a** to be transmitted; and pressing the sequence *SHIFT A* causes the binary code B´01000001´ for the character **A** to be transmitted. Bit 6 is forced low to define an uppercase character.

When CTRL is pressed simultaneously with one of keys A-Z (uppercase only), the space bar, or the special character keys {, [,], }, or ?, an ASCII control code is generated. For example, the *CTRL Z* keyboard sequence causes the binary code B´00011010´ to be generated. The only difference between this code and the binary code for Z (B´01011010´) is that bit 7 is forced low to define the control code.

When the SHIFT and CTRL keys are pressed simultaneously, the shift function is selected in most cases. The only exceptions occur with the { and ? keys. The *SHIFT CTRL* { sequence causes the control character RS (B′00011110′) to be transmitted. The *SHIFT CTRL ?* sequence causes the control character **US** (B′00011111′) to be transmitted.

When the REPT key is locked down, the auto-repeat feature is enabled on all keys except: F1 - F12, HARD_COPY, SETUP, GRAPH, CLEAR_HOME, LINE_LOCAL, TERM, CAPS_LOCK, CTRL, SHIFT (both keys), RETURN, and all numeric pad keys. When any other key is held down with the keyboard in auto repeat mode, repeated character transmission occurs. The initial rate is less than 2 Hz, but this increases to about 11 Hz in less than two seconds. Pressing the REPT key a second time causes it to release upwards, canceling the auto repeat feature.

Pressing the CAPS_LOCK key causes it to assume a locked-down position, asserting the "caps lock" function. This is actually a limited shift operation that applies to the alphabetic (A-Z) keys only. Alphabetic keys struck while the keyboard is in "caps lock" mode generate uppercase characters. Pressing the CAPS_LOCK a second time causes it to release upward, canceling the "caps lock" mode.

2.2.2 Alphabetic Keys

The alphabetic keys are used to produce uppercase and lowercase ASCII displayable character codes, and ASCII control codes. Table 13A-2 shows the code and character produced when each key is pressed alone, with the SHIFT key, or with the CTRL key. The code in the table is shown in hexadecimal notation.

Table 13A-2. Alphabetic Key Codes

KEY LABEL	KEY ALONE		SHIFT+KEY		CTRL+KEY	
	CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER
A	X'61'	a	X'41'	A	X'01'	SOH
B	X'62'	b	X'42'	B	X'02'	STX
C	X'63'	c	X'43'	C	X'03'	ETX
D	X'64'	d	X'44'	D	X'04'	EOT
E	X'65'	e	X'45'	E	X'45'	ENQ
F	X'66'	f	X'46'	F	X'06'	ACK
G	X'67'	g	X'47'	G	X'07'	BEL
H	X'68'	h	X'48'	H	X'08'	BS
I	X'69'	i	X'49'	I	X'09'	HT
J	X'6A'	j	X'4A'	J	X'0A'	LF
K	X'6B'	k	X'4B'	K	X'0B'	VT
L	X'6C'	l	X'4C'	L	X'0C'	FF
M	X'6D'	m	X'4D'	M	X'0D'	CR
N	X'6E'	n	X'4E'	N	X'0E'	SO
O	X'6F'	o	X'4F'	O	X'0F'	S
P	X'70'	p	X'50'	P	X'10'	DLE
Q	X'71'	q	X'51'	Q	X'11'	DC1
R	X'72'	r	X'52'	R	X'12'	DC2
S	X'73'	s	X'53'	S	X'13'	DC3
T	X'74'	t	X'54'	T	X'14'	DC4
U	X'75'	u	X'55'	U	X'15'	NAK
V	X'76'	v	X'56'	V	X'16'	SYN
W	X'77'	w	X'57'	W	X'17'	ETB
X	X'78'	x	X'58'	X	X'18'	CAN
Y	X'79'	y	X'59'	Y	X'19'	EM
Z	X'7A'	z	X'5A'	Z	X'1A'	SUB

2.2.3 Standard Numeric Keys

The standard numeric keys generate ASCII displayable numbers and symbols. The CTRL key is ignored when used with these keys. Table 13A-3 shows the code and character produced when each key is pressed alone, with the SHIFT key, or with the CTRL key. The code in the table is shown in hexadecimal notation.

Table 13A-3. Standard Numeric Key Codes

KEY	KEY ALONE		SHIFT+KEY		CTRL+KEY	
LABEL	CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER
0	X´30´	0	X´29´	)	X´30´	0
1	X´31´	1	X´21´	!	X´30´	1
2	X´32´	2	X´40´	@	X´32´	2
3	X´33´	3	X´23´	#	X´33	3
4	X´34´	4	X´24´	$	X´34´	4
5	X´35´	5	X´25´	%	X´35´	5
6	X´36´	6	X´5E´	^	X´36´	6
7	X´37´	7	X´26´	&	X´37´	7
8	X´38´	8	X´2A´	*	X´38´	8
9	X´39´	9	X´28´	(	X´39´	9

2.2.4 Special Character Keys

The special character keys are detailed in Table 13A-4. The code in the table is shown in hexadecimal notation. These keys can be pressed alone, with the SHIFT key, and with the CTRL key. Note the varying response given to the CTRL key; in some instances, the unshifted key character is produced. In other cases, a control character is generated. In two cases, X´1F´ and X´1E´, both the SHIFT and CTRL keys must be used with the special character key to produce the control code shown in Table 13A-4.

Table 13A-4. Special Character Key Codes

KEY LABEL	KEY ALONE CODE	KEY ALONE CHARACTER	SHIFT+KEY CODE	SHIFT+KEY CHARACTER	CTRL+KEY CODE	CTRL+KEY CHARACTER
— -	X′2D′	- (minus)	X′5F′	— (underline)	X′2D′	- (minus)
+ =	X′3D′	=	X′2B′	+	X′2B′	=
~ \`	X′60′	\`	X′7E′	~	X′1E′	RS*
{ [	X′5B′	[	X′7B′	{	X′1B′	ESC
}]	X′5D′	]	X′7D′	}	X′1D′	GS
\	X′5C′	\	X′7C′	|	X′1C′	FS
: ;	X′3B′	;	X′3A′	:	X′3B′	;
" ′	X′27′	′	X′22′	"	X′27′	′
< ,	X′2C′	,	X′3C′	<	X′2C′	,
> .	X′2E′	.	X′3E′	>	X′2E′	.
? /	X′2F′	/	X′3F′	?	X′1F′	US*

*These control codes may also be produced by pressing both SHIFT and CTRL in conjunction with the indicated key.

2.2.5 Terminal Function Keys

The terminal function keys produce codes used by a typical video display terminal. These keys enable an operator to generate any commonly used terminal control character with a single keystroke. Table 13A-5 lists the codes and characters generated by the terminal function keys. The code in the table is shown in hexadecimal notation.

The codes produced by these keys are identical to those generated by the conventional two-key control sequences described in Table 13A-5.

The SHIFT and CTRL keys have no effect on the codes produced by the terminal function keys, except for the *CTRL Space_Bar* sequence that generates an ASCII NUL character.

Table 13A-5. Terminal Function Key Codes

KEY LABEL	KEY ALONE		SHIFT+KEY		CTRL+KEY	
	CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER
BACKSPACE	X'08'	BS	X'08'	BS	X'08'	BS
DEL	X'7F'	DEL	X'7F'	DEL	X'7F'	DEL
RETURN	X'0D'	CR	X'0D'	CR	X'0D'	CR
LINE_FEED	X'0A'	LF	X'0A'	LF	X'0A'	LF
ESC	X'1B'	ESC	X'1B'	ESC	X'1B'	ESC
TAB	X'09'	HT	X'09'	HT	X'09'	HT
(none)	X'20'	(space)	X'20'	(space)	X'00'	NUL

2.2.6 Function Keys

Table 13A-6 illustrates the codes produced by each function key as it is used individually, or in combination with the SHIFT and/or CTRL keys. The code in the table is shown in hexadecimal notation. Each transmitted code is preceded by X′16′.

Table 13A-6. Function Key Codes

KEY LABEL	KEY ALONE		SHIFT+KEY		CTRL+KEY	
	CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER
F1	X′61′	a	X′41′	A	X′01′	SOH
F2	X′62′	b	X′42′	B	X′02′	STX
F3	X′63′	c	X′43′	C	X′03′	ETX
F4	X′64′	d	X′44′	D	X′04′	EOT
F5	X′65′	e	X′45′	E	X′05′	ENQ
F6	X′66′	f	X′46′	F	X′06′	ACK
F7	X′67′	g	X′47′	G	X′07′	BEL
F8	X′68′	h	X′48′	H	X′08′	BS
F9	X′69′	i	X′49′	i	X′09	HT
F10	X′6A′	j	X′4A′	J	X′0A′	LF
F11	X′6B′	k	X′4B′	K	X′0B′	VT
F12	X′6C′	l	X′4C′	L	X′0C′	FF

Note: All codes are preceded by X′16′.

2.2.7 Numeric/Application Mode Keys

Table 13A-7 illustrates the codes and characters produced by the numeric/ application mode keys. The code in the table is shown in hexadecimal notation. Neither SHIFT or CTRL affects the ENTER key, and no codes are modified by the CTRL key.

Table 13A-7. Numeric/Application Mode Key Codes

KEY LABEL	KEY ALONE		SHIFT+KEY		CTRL+KEY	
	CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER
0	X′30′	0	X′29′	)	X′30′	0
1	X′31′	1	X′21′	!	X′31′	1
2	X′32′	2	X′40′	@	X′32′	2
3	X′33′	3	X′23′	#	X′33′	3
4	X′34′	4	X′24′	$	X′34′	4
5	X′35′	5	X′25′	%	X′35′	5
6	X′36′	6	X′5E′	^	X′36′	6
7	X′37′	7	X′26′	&	X′37′	7
8	X′38′	8	X′2A′	*	X′38′	8
9	X′39′	9	X′28′	(	X′39′	9
.	X′2E′	.	X′3E′	>	X′2E′	.
,	X′2C′	,	X′3C′	<	X′2C	,
-	X′2D′	(minus) -	X′5F′	(underline) —	X′2D′	(minus) -
ENTER	X′OD′	CR	X′OD′	CR	X′OD′	CR

Note: All codes are preceded by X′16′.

2.2.8 Device Control Keys

Table 13A-8 illustrates the codes and characters produced by the device control keys. The codes produced by these keys are modified by SHIFT and CTRL.

Table 13A-8. Device Control Key Codes

KEY LABEL	KEY ALONE		SHIFT+KEY		CTRL+KEY	
	CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER
HARD COPY	X′6E′	n	X′4E′	N	X′0E′	SO
SETUP	X′6F′	o	X′4F′	O	X′0F′	SI
GRAPH	X′70′	p	X′50′	P	X′10′	DLE
CLEAR HOME	X′71′	q	X′51′	Q	X′11′	DC1
LINE LOCAL	X′72′	r	X′52′	R	X′12′	DC2
TERM	X′73′	s	X′53′	S	X′13′	DC3
←	X′77′	w	X′57′	W	X′17′	ETB
→	X′78′	x	X′58′	X	X′18′	CAN
↑	X′79′	y	X′59′	Y	X′19′	EM
↓	X′7A′	z	X′5A′	Z	X′1A′	SUB

```
The Cursor Up key becomes Scroll Up when shifted.
The Cursor Down key becomes Scroll Down when shifted.

Note: All codes are preceded by X′16′.
```

2.3 Keyboard LED Display

The keyboard LED display will recognize and display the following ASCII characters:

```
!"#$%&'()*+,-./0123456789:;

?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_
```

In addition to the above characters, *CTRL E, CTRL G, CTRL V,* BACKSPACE, DEL, RETURN, space, and lowercase alphabetic characters are recognized.

Lowercase alphabetic characters are converted to uppercase and displayed. *CTRL E* causes the keyboard to send the following message to the PS 390:

```
KBxxxD
```

where xxx is the PROM version number in the keyboard.

CTRL G generates a bell tone. *CTRL V*, BACKSPACE, DEL, and RETURN are used as described below. All other characters are ignored.

2.4 Keyboard Display Modes

The keyboard display operates in two modes:

- Line mode
- Function key label mode

2.4.1 Line Mode

In line mode, the LEDs fill from left to right as characters for display are received. A left-justified line up to 96 characters long (including spaces) can be displayed.

The DEL and BACKSPACE characters are processed only in line mode. The BACKSPACE character causes the entire display to logically move left one LED display position. The DEL character causes the most recently entered character to be deleted.

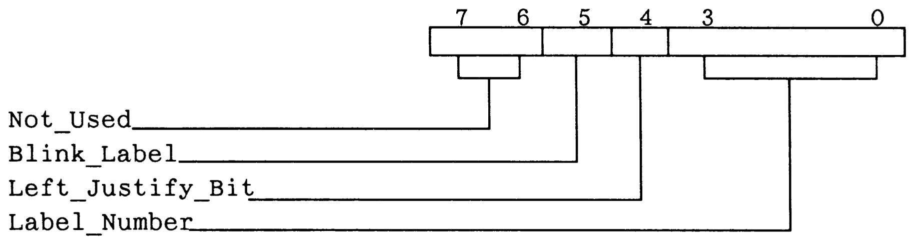
All data transmitted to the LEDs for display in line mode must be terminated with a RETURN character. After a RETURN character is entered, the display is cleared when the next valid character is received. The received character is output to the leftmost LED character, and the LEDs are filled left to right as before.

2.4.2 Function Key Label Mode

Function key label mode is used to provide a descriptive label for each function key. The data input to the keyboard for this purpose must conform to the following format:

X′16′	Label Parameter Byte	0 to 8 Characters	RETURN

The label parameter byte specifies blinking, left justification, and label number; its format is as follows:



Note

Label values 0–11 correspond to function key numbers 1–12.

When the blink label bit is 1, the characters in the label location (0 - 11) specified in bits 0–3 blinks. When this bit is 0, the segment does not blink. To blink or unblink an existing label, it is only necessary to send X′16′, the label parameter byte, and a RETURN.

When the left justify bit is 1, the function key label is left-justified in the specified label location; spaces are placed in any unused characters. When this bit is 0, the label is automatically centered in the segment location.

The "0 to 8 Characters" specified in the above label format constitute actual ASCII characters to display. The RETURN is a required terminator that must appear following each LED label string.

To describe function key label mode, examples of function key labels and the data required to produce and modify them are provided below.

1. To center an "unblinking" label X AXIS over key F6 the following hexadecimal string is used:

Byte	Meaning
X´16´	CTRL V
X´05´	Don´t Blink; Center; use Segment 5
X´58´	X
X´20´	Space
X´41´	A
X´58´	X
X´49´	I
X´53´	S
X´OD´	RETURN

2. To make the existing label blink, the following hexadecimal string is used:

Byte	Meaning
X´16´	CTRL V
X´25´	Blink; Center; use Segment 5
X´OD´	RETURN

3. The following hexadecimal string is used to "unblink" the existing label:

Byte	Meaning
X´16´	CTRL V
X´05´	Don´t Blink; Center; use Segment 5
X´OD´	RETURN

4. To code the label Y TRANS for presentation over key F12 with the label left-justified and blinking, use the following hexadecimal string:

Byte	Meaning
X'16'	CTRL V
X'3B'	Blink; Left-justify; Use Segment 11
X'59'	Y
X'20'	Space
X'54'	T
X'52'	R
X'41'	A
X'4E'	N
X'53'	S
X'0D'	RETURN

3. Control Dials Unit

The control dials unit is a modular interactive device that is microprocessor controlled. Power, ground, and communication lines are routed through a modular phone cord from the peripheral multiplexer to the control dials interface card. It uses a single, eight-conductor flexible interface cable with locking modular plugs. The dials are used to communicate dynamic, incrementing, and decrementing data to the PS 390. There is an effective resolution of 1024 counts per turn.

3.1 Operating Modes

The control dials unit operates in the following modes.

- Message

  The control dials unit outputs rotational values in message mode only when enabled to do so by a setup command from the JCP. Each dial is individually programmable. The message mode may be entered any time after initial power-up and is entirely under the control of the PS 390.

- LED Label Mode

  This mode allows each eight-character LED label to be individually defined.

3.2 Operation

The control dials unit outputs pulses when any of the dials are turned. From the pulses, the control dials interface determines:

- Which dial is being turned.
- What direction the dial is turning.
- How far the dial is rotated. Dial position is evaluated in terms of the number of changes of delta.

After the control dials interface analyzes dial motion, rotational information is transmitted to the JCP. The following paragraphs describe data formats and codes exchanged in dial and LED display operation.

Two messages set up the operating mode for the control dials unit. One command specifies the minimum rotation count delta required before a sample is output to the PS 390, and the other command specifies the maximum rate at which the control dials unit sends a new delta update to the PS 390. The control dials unit outputs relative delta values only; that is, the position of each dial is reported in terms of its last sampled location. These inputs can come from the initial function instance DSET1...DSET8, and must be done before any output can occur after power-up.

The message to specify the rotation count delta for a particular control dial consists of the following four-byte sequence:

X'16'	Control Byte	MSB	LSB

The control byte specifies the dial number in the following format:

```
1x0xxnnn
```

where the n's specify the dial number between 000 and 111 (0 - 7.), and the x's may be either zero or one.

The most significant byte (MSB) and least significant byte (LSB) together specify the 16-bit delta value. This number may be any value between 1 and 65535; use of negative or zero values is not recommended.

The message that specifies the maximum update in seconds is in the following four-byte format:

X'16'	Control byte	Reserved	Time Count

The control byte is in the following format:

```
1x1xxxxx
```

where all x's may be either zero or one. This means that the specified maximum update applies to all dials.

The next byte is reserved for possible future use.

The final byte consists of a binary number that specifies the sample time value. The following sample times are available:

Hex	Decimal	Updates/Sec.
05	5	60
0A	10	30
1E	30	10

3.2.1 Dial Setup Programming Examples

To specify a maximum update rate of 10 updates per second:

Byte	Meaning
X'16'	CTRL V
X'90'	Setup maximum update rate
X'00'	Reserved byte
X'1E'	10 updates per second (decimal 30)

To set dial four for the minimum rotation count delta required before a sample is output:

Byte	Meaning
X'16'	CTRL V
X'84'	Setup Dial 4
X'00'	Delta MSB
X'06'	Delta LSB

The data format that is output from the control dials unit takes the following form:

X′16′	Dial Number	Sample MSB	Sample LSB

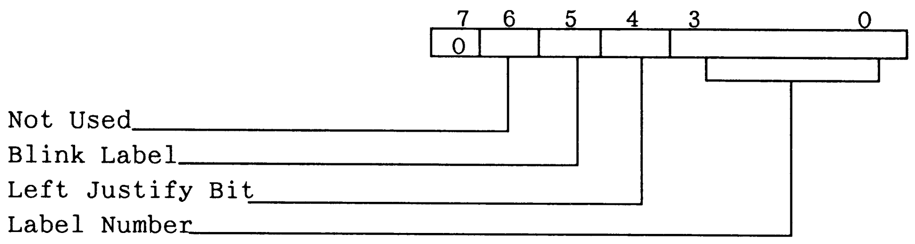
3.3 LED Display Operation

The control dials unit has eight 8-character LED displays. Each display functions as a label for a dial. The LED displays are much like those on the keyboard, displaying the same characters and responding to the same codes. The control dials unit LEDs operate in label (segment) mode. That is, each display is separately programmed and functions independently of the other LEDs.

The LED label message format is as follows:

X′16′	Control Byte	0 to 8 Characters

The X′16′ character indicates the beginning of a command string. The control byte specifies blinking, left justification, and LED label number.



The control byte format is as follows:

- Bit 7 in the control byte is always 0.
- Bit 6 is not used.

When the blink label bit (bit 5) is 1, the label blinks. When this bit is 0, the label does not blink.

When the left justify bit (bit 4) is 1, the label is left-justified in the specified label location. When this bit is 0, the label is automatically centered in the label location.

The label number bits (bits 3–0) specify the LED label location (0–7).

The "0 to 8 characters" are the ASCII characters to be displayed on the selected LED label. If there is no label message (character count = 0), then the current message in the LED label is set up according to the values of bit 5 in the control byte (that is, the LED will blink or not blink).

4. Function Buttons Unit

The function buttons unit gives an expanded capability for program selection, providing 32 programmable function buttons in addition to the 12 function keys on the keyboard. Power and communications for the function buttons unit are provided through a single modular phone cord that connects to the peripheral multiplexer. The function buttons are lighted by incandescent bulbs. As with the function keys on the keyboard, pressing a function button results in a user-specified action.

The function buttons unit is arranged with one row of four buttons, four rows of six buttons, and a final row of four buttons. The buttons are numbered from left to right, beginning at the top row of four buttons, with the first button labeled 0. Buttons can be programmed from the PS 390 to light when activated and go out when not activated.

During operation, the function buttons respond to valid characters from the PS 390 and send a character to the PS 390 if a button is pressed. Inputs to the PS 390 from the function buttons unit are sent to the appropriate function network which determines the button functions. The activity of the lights backing the buttons is determined by messages sent from the PS 390 to the function buttons unit.

4.1 Communications Protocol

During operation, the PS 390 and the function buttons unit use the communications protocol outlined below.

NOTE

The displayed messages (such as X′05′, Ctrl E) in this section show both the ASCII equivalent (X′05′) and the actual character (Ctrl E). When "KEY" is entered as the actual character, it indicates the key entered by the user.

Turn ON All Lights Message (From PS 390)

This message from the PS 390 turns ON all 32 lights in the function buttons unit:

```
<SI>,X'0F',Ctrl O
```

Turn OFF All Lights Message (From PS 390)

This message from the PS 390 turns OFF all 32 lights in the function buttons unit:

```
<SO>,X'0E',Ctrl N
```

Turn ON Light KEY Message (From PS 390)

This message from the PS 390 turns ON one of the 32 lights in the function buttons unit (no other lights are affected):

```
(X'40'+ KEY)
```

The value chosen for KEY (which should be a hex number from [X'00'] to [X'1F']) determines the specific light selected. If the designated light is already ON, this message has no affect.

Turn OFF Light KEY Message (From PS 390)

This message from the PS 390 turns OFF one of the 32 lights in the function buttons unit (no other lights are affected):

```
(X'60'+ KEY)
```

The value chosen for KEY (which should be a hex number from [X'00'] to [X'1F']) determines the specific light selected. If the designated light is already OFF, this message has no affect.

Key Down, Light ON Message (From Buttons)

This message from the function buttons unit reports to the PS 390 that a KEY has been pressed down and that the status of the light in that KEY is ON:

```
(X'40'+ KEY)
```

The value of KEY should be a number (X'00') to (X'1F') corresponding to one of the 32 keys in the function buttons unit.

Key Down, Light OFF Message (From Buttons)

This message from the function buttons unit reports to the PS 390 that a KEY has been pressed down and that the status of the light in that KEY is OFF:

```
(X'60'+ KEY)
```

The value of KEY should be a number (X'00') to (X'1F') corresponding to one of the 32 keys in the function buttons unit.

5. Data Tablet

There are two data tablets available for use with the PS 390. Both tablets are identical for the PS 300 style and the PS 390 style interactive devices. There is a 6-inch by 6-inch and a 12-inch by 12-inch tablet, each with a four-button puck. Both are alike, except for their active areas, and both provide digitizing and picking functions for the PS 390.

5.1 Operating Modes

Data tablet modes may be controlled externally under program control. The following operating modes are available:

- Point mode

  Pressing a puck button at a given tablet location causes one X,Y coordinate pair (sample) to be transmitted.

- Stream mode

  X,Y coordinate pairs are generated continuously at the selected sampling rate when the puck is near the active area of the tablet.

- Switched stream mode

  Pressing a button on the puck causes X,Y coordinate pairs to be output continuously at the selected sampling rate until the button is released.

Both the mode and the sampling rate may be changed under program control from the PS 390 by sending the data tablet an ASCII character. Table 13A-9 lists the ASCII codes.

Table 13A-9. Binary Data Transmission Codes

Mode	Binary Rate	Uppercase ASCII Character
Stop	-	S
Point	-	P
Switched Stream	2	@
	4	A
	10	B
	20	C
	35	D
	70	E
	141	F
	141	G
Stream	2	H
	4	I
	10	J
	20	K
	35	L
	70	M
	141	N
	141	O

5.1.1 Binary Data Format

The binary formatted RS-232 interface is a five-byte count output. Binary format is shown in Table 13A-10.

Table 13A-10. Data Tablet Binary Format

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	P	1	F3	F2	F1	F0	0	0
2	P	0	X5	X4	X3	X2	X1	X0
3	P	0	X11	X10	X9	X8	X7	X6
4	P	0	Y5	Y4	Y3	Y2	Y1	Y0
5	P	0	Y11	Y10	Y9	Y8	Y7	Y6

6. The Optical Mouse

The optical mouse transforms position information into a digital form acceptable to the PS 390. The optical mouse uses a three-button mouse unit in conjunction with a reflective pad to provide X- and Y-axis position information.

The mouse uses LEDs reflecting off the pad to provide directional information to the control logic in the mouse. This movement is then translated into relative X and Y movement information. The data is transmitted serially to the PS 390 through the peripheral multiplexer.

NOTE

The optical mouse pad must be oriented horizontally to the user for proper mouse operation. Furthermore, the mouse cord (tail) should lead away from the user.

6.1 Protocol

The mouse protocol is 9600 baud asynchronous serial with one start bit, one stop bit, and eight data bits. The least significant data bit is transmitted first. Blocks of five bytes are sent whenever there is a change of mouse state (switches or position) since the last transmission. The protocol is as follows:

1. Byte 1: Bits 3 through 7 represent the sync for the start of the data block with bit 7 = 1 and bits 3–6 = 0. Bits 0 through 2 define switch status (0 switches the depressed state). With the mouse oriented so that the cord is facing away from the user, the right switch status is indicated by bit 0, the middle switch status by bit 1, and the left switch status is indicated by bit 2.

2. Byte 2: Bits 0 through 7 represent the incremental change in the X-direction since the last complete report up to the time Byte 1 starts transmission. The data is in the two's complement form and has a value limit of +/- 127. With the mouse cord facing away from the user, moving the mouse to the right produces positive X values and moving the mouse to the left produces negative X values.

3. Byte 3: Bits 0 through 7 represent the incremental change in the Y-direction since the last complete report up to the time Byte 1 starts transmission. The data is in the two's complement form and has a value limit of +/- 127. Moving the mouse towards its mouse cord produces positive X values and moving the mouse away from its cord produces negative Y values.

4. Byte 4: Bits 0 through 7 follow the same format as Byte 2 and represent the data acquired since the beginning of Byte 1 transmission.

5. Byte 5: Bits 0 through 7 follow the same format as Byte 3 and represent the data acquired since the beginning of Byte 4 transmission.

Table 13A-11. Mouse Bit Protocol

	MSB							LSB
Bit No.	7	6	5	4	3	2	1	0
Byte 1	1	0	0	0	0	L	M	R
Byte 2	X7	X6	X5	X4	X3	X2	X1	X0
Byte 3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
Byte 4	X7	X6	X5	X4	X3	X2	X1	X0
Byte 5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

RM13B. INTERACTIVE DEVICES

PS 390 STYLE

CONTENTS

ILLUSTRATIONS

TABLES

Section RM13B
Interactive Devices
PS 390 Style

Two sets of interactive devices are available with the PS 390: the PS 300-style devices and the PS 390-style devices. The interactive devices from the two styles cannot be mixed with the exception of the data tablets, and the optical mouse, which are common to both styles.

The PS 390-style interactive devices include:

- Keyboard without LEDs
- Control dials unit without LEDs
- Function buttons unit
- Data tablet (6x6 or 12x12) with puck
- Optical mouse

The light pen is not supported on the PS 390.

The PS 390 interfaces with the interactive devices through the peripheral multiplexer which supplies the power to the interactive devices and serves as their input/output path to the PS 390. The peripheral multiplexer combines the signals from the interactive devices and transmits them to the PS 390.

This section describes the PS 390-style interactive devices and peripheral multiplexer. Section *RM13A* describes the PS 300-style interactive devices and peripheral multiplexer.

1. The Peripheral Multiplexer

The peripheral multiplexer serves as the connection point between the PS 390 system and the interactive devices. It provides power to the interactive devices and combines their signals and transmits them to the PS 390. It also routes any signals which the the system may send back to the appropriate interactive device.

The peripheral multiplexer is housed in a metal box which fits beneath the raster display pedestal. The interactive devices connect to the five connectors on the front of the multiplexer. Each connector is uniquely dedicated to a specific interactive device.

The peripheral multiplexer provides programmed logic which allows the data from the interactive devices to be multiplexed over a single RS-232C line into the controller via Port 5 on the rear of the PS 390.

1.1 Functional Characteristics

The peripheral multiplexer consists of a circuit card which is connected to five input ports and one output port. The five input ports support the following interactive devices:

- Keyboard
- Control dials unit
- Function buttons unit
- Data tablet (6 by 6 or 12 by 12) with puck
- Optical mouse

Figure 13B-1 shows the backside connectors and plugs for the peripheral multiplexer. Figure 13B-2 shows the peripheral connections for the PS 390-style peripheral set.

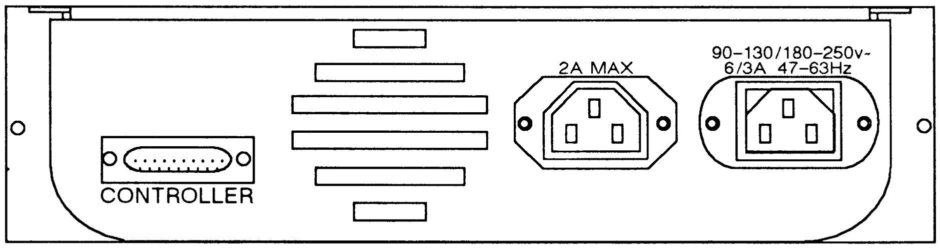


Figure 13B-1. Backside Connectors for the Peripheral Multiplexer

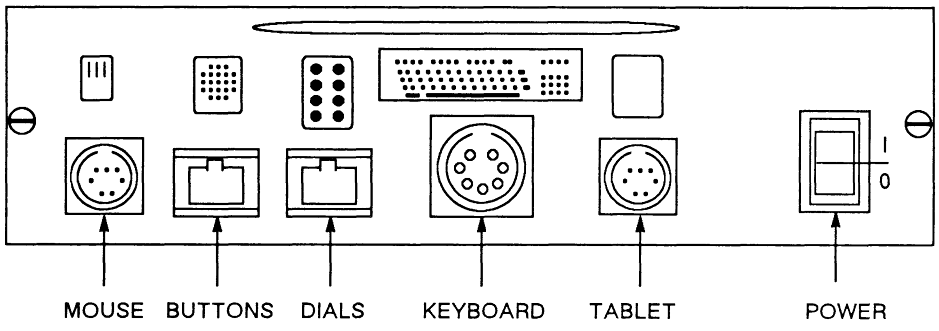


Figure 13B-2. Connectors for the PS 390 Style Interactive Devices

1.2 Data Framing and Transmission Rates

The data sent to and from the peripheral multiplexer is asynchronous data with each byte containing eight data bits with no parity, one start bit and one stop bit. The data transmission rate of the peripheral multiplexer to and from the PS 390 is 19,200 baud. The transmission rates between the interactive devices and the peripheral multiplexer are shown in Table 13B-1.

Table 13B-1. Interactive Device Transmission Rates

Device	**Baud Rate**
`Keyboard Port x'B1'`	`1200 Baud`
`Control Dials Port x'B2'`	`9600 Baud`
`32 Func. Buttons Port x'B3'`	`9600 Baud`
`Mouse Port x'B4'`	`9600 Baud`
`Data Tablet Port x'B6'`	`9600 Baud`

2. The PS 390 Keyboard

The PS 390 Keyboard's main function is to generate and transmit ASCII displayable characters, ASCII control characters, and PS 390 system sequences.

The PS 390 keyboard must plug into the peripheral multiplexer which supports the PS 390 peripheral set.

The keyboard measures 19.76 inches (50.19 cm) long by 8.26 inches (20.98 cm) deep. The keyboard stands 1.40 inches (3.56 cm) high on four rubber pads.

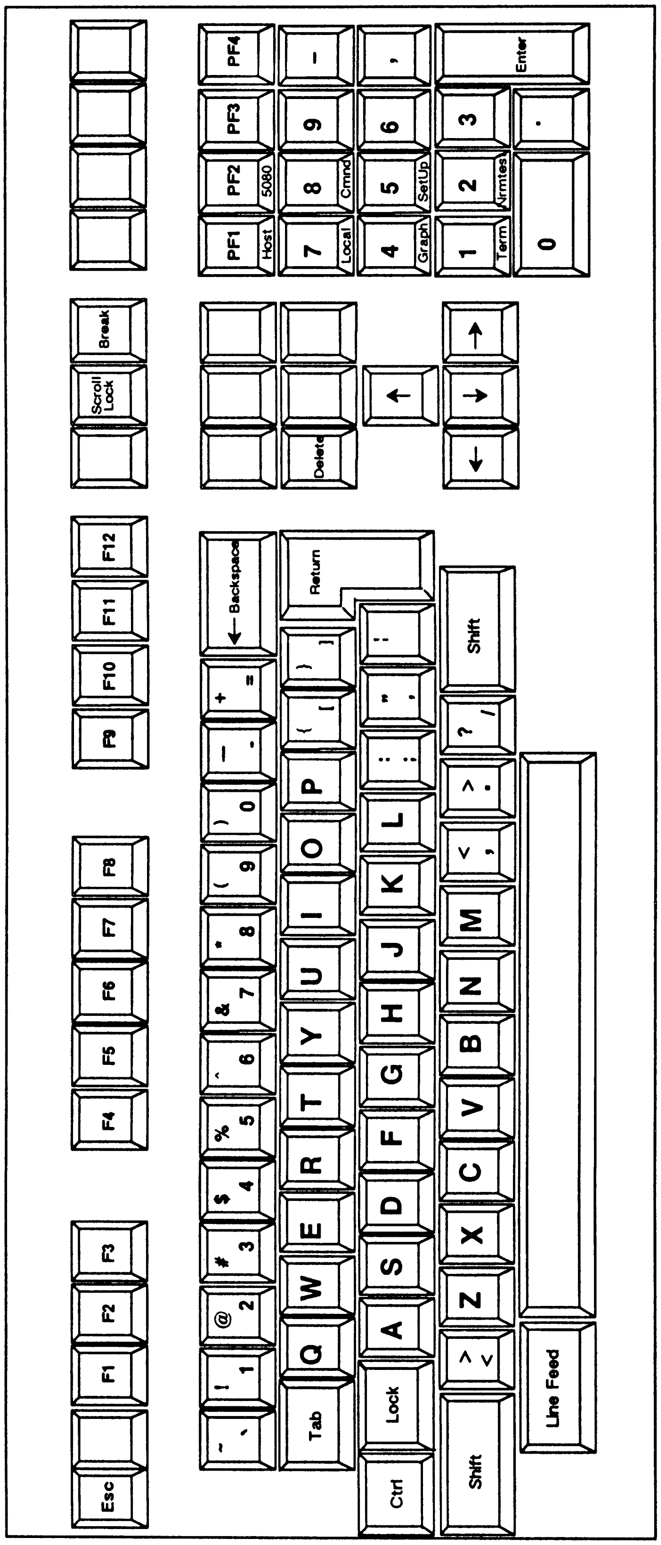


Figure 13B-3. The PS 390 Style Keyboard

2.1 Interface Cable

The Interface Cable is a 5-conductor, flexible cable with a shielded DIN plug which connects the PS 390 Keyboard to the front of the Peripheral Multiplexer. The cable may be stretched to permit many different work station arrangements.

2.2 Keyboard Operation

The PS 390 Keyboard allows the operator to input ASCII characters and other sequences to the Joint Control Processor by means of a typewriter-like keyboard. Keyboard operation is discussed in detail in the following paragraphs.

2.2.1 Data Entry

The keys fall into eight general categories:

- Keyboard Function Control
- Alphabetic
- Standard Numeric
- Special Character
- Terminal Function
- PS 390 Function
- Numeric/Application Mode
- PS 390 Device Control

NOTE

When instructions are given to press two or more keys simultaneously, the key sequence will be shown in italics. For example, *CTRL V* means that the CTRL and V keys are pressed simultaneously.

The following is a detailed description of the eight general key categories.

2.2.2 Keyboard Function Control Keys

The Keyboard Function Control keys are unencoded, local controls. No codes are transmitted when these keys are pressed individually or in combination with each other.

The Keyboard Function Control keys are as follows:

- Shift Key (2)
- CTRL (Control) Key

The Keyboard Function Control keys are used to modify the codes transmitted by other keys, as follows:

- When either SHIFT key is pressed simultaneously with a displayable character key, the uppercase code for that key is generated. If the key does not have an uppercase function, the SHIFT key is ignored. For example, striking the **A** key causes the code B´01100001´ for the character **a** to be transmitted; the sequence *SHIFT A* causes the code B´01000001´ for the character **A** to be transmitted. Note that bit 6 is forced low to define an uppercase character.
- When CTRL is pressed simultaneously with one of keys A-Z (uppercase only), the space bar, or the Special Character keys , [,], |, , or ?, an ASCII control code is generated. For example, the *CTRL Z* keyboard sequence causes the code B´00011010´ to be generated. Note that the only difference between this code and that for **Z** (B'010 11010') is that bit 7 is forced low to define the control code.

When the SHIFT and CTRL keys are pressed simultaneously, the CTRL function is selected in most cases. The only exceptions occur with the ~ and / keys. *SHIFT CTRL ~* causes the control character **RS** (B´00011110´) to be transmitted. *SHIFT CTRL /* causes the control character **US** (B´00011111´) to be transmitted. The auto-repeat feature is enabled on all keys except: F1 - F12, SETUP, GRAPH, HOST, CMND, LOCAL, TERM, LOCK, CTRL, SHIFT (both keys), RETURN, and all numeric pad keys. When any other key is held down, repeated character transmission occurs. The rate is 15 +/- 2 Hz.

Pressing the LOCK key enables the "shift lock" function. This is a shift operation that applies to all keys. Pressing either of the two shift keys causes the "shift lock" mode to be disabled.

2.2.3 Alphabetic Keys

The Alphabetic Keys are used to produce uppercase and lowercase ASCII displayable character codes and ASCII control codes. Table 13B-2 shows the code and character produced when each key is pressed alone, with the SHIFT key, or with the CTRL key.

Table 13B-2. Alphabetic Key Codes

Key	*Key Alone*		*SHIFT+Key*		*CTRL+Key*	
Label	*Code*	*Char*	*Code*	*Char*	*Code*	*Char*
A	X'61' 97	a	X'41' 65	A	X'01' 1	SOH
B	X'62' 98	b	X'42' 66	B	X'02' 2	STX
C	X'63' 99	c	X'43' 67	C	X'03' 3	ETX
D	X'64' 100	d	X'44' 68	D	X'04' 4	EOT
E	X'65' 101	e	X'45' 69	E	X'45' 5	ENQ
F	X'66' 102	f	X'46' 70	F	X'06' 6	ACK
G	X'67' 103	g	X'47' 71	G	X'07' 7	BEL
H	X'68' 104	h	X'48' 72	H	X'08' 8	BS
I	X'69' 105	i	X'49' 73	I	X'09' 9	HT
J	X'6A' 106	j	X'4A' 74	J	X'0A' 10	LF
K	X'6B' 107	k	X'4B' 75	K	X'0B' 11	VT
L	X'6C' 108	l	X'4C' 76	L	X'0C' 12	FF
M	X'6D' 109	m	X'4D' 77	M	X'0D' 13	CR
N	X'6E' 110	n	X'4E' 78	N	X'0E' 14	SO

Table 13B-2. Alphabetic Key Codes (cont.)

Key	*Key Alone*		*SHIFT+Key*		*CTRL+Key*	
Label	*Code*	*Char*	*Code*	*Char*	*Code*	*Char*
O	X'6F' 111	o	X'4F' 79	O	X'0F' 15	SI
P	X'70' 112	p	X'50' 80	P	X'10' 16	DLE
Q	X'71' 113	q	X'51' 81	Q	X'11' 17	DC1
R	X'72' 114	r	X'52' 82	R	X'12' 18	DC2
S	X'73' 115	s	X'53' 83	S	X'13' 19	DC3
T	X'74' 116	t	X'54' 84	T	X'14' 20	DC4
U	X'75' 117	u	X'55' 85	U	X'15' 21	NAK
V	X'76' 118	v	X'56' 86	V	X'16' 22	SYN
W	X'77' 119	w	X'57' 87	W	X'17' 23	ETB
X	X'78' 120	x	X'58' 88	X	X'18' 24	CAN
Y	X'79' 121	y	X'59' 89	Y	X'19' 25	EM
Z	X'7A' 122	z	X'5A' 90	Z	X'1A' 26	SUB

2.2.4 Standard Numeric Keys

The shiftable Standard Numeric keys are similar to the shiftable numeric/ symbol keys that appear on a typewriter; they generate ASCII displayable numbers and symbols. The CTRL key is ignored when used with these keys. Table 13B-3 shows the code and character produced when each key is pressed alone, with the SHIFT key, or with the CTRL key.

Table 13B-3. Standard Numeric Keys Codes

Key	*Key Alone*		*SHIFT+Key*		*CTRL+Key*	
Label	*Code*	*Char*	*Code*	*Char*	*Code*	*Char*
0	X'30' 48	0	X'29' 41	)	X'30' 48	0
1	X'31' 49	1	X'21' 33	!	X'31' 49	1
2	X'32' 50	2	X'40' 64	@	X'32' 50	2
3	X'33' 51	3	X'23' 35	#	X'33' 51	3
4	X'34' 52	4	X'24' 36	$	X'34' 52	4
5	X'35' 53	5	X'25' 37	%	X'35' 53	5
6	X'36' 54	6	X'5E' 94	∧	X'36' 54	6
7	X'37' 55	7	X'26' 38	&	X'37' 55	7
8	X'38' 56	8	X'2A' 42	*	X'38' 56	8
9	X'39' 57	9	X'28' 40	(	X'39' 57	9

2.2.5 Special Character Keys

The shiftable Special Character keys are used to produce both ASCII displayable characters and ASCII control characters. Table 13B-4 shows the codes and characters produced when these keys are activated alone, with the SHIFT key, and with the CTRL key. Note the varying response given to the CTRL key; in some instances, the unshifted key character is produced. In other cases, a control character is generated.

Table 13B-4. Special Character Keys

Key	*Key Alone*		*SHIFT+Key*		*CTRL+Key*	
Label	*Code*	*Char*	*Code*	*Char*	*Code*	*Char*
_ -	X’2D’ 45	- (minus)	X’5F’ 95	— (underline)	X’2D’ 45	- (minus)
+ =	X’3D’ 61	=	X’2B’ 43	+	X’3D’ 61	=
~ `	X’60’ 96	~	X’7E’ 126	`	X’1E’ 30	RS
{ [	X’5B’ 91	[	X’7B’ 123	{	X’1B’ 27	ESC
}]	X’5D’ 93	]	X’7D’ 125	}	X’1D’ 29	GS
| \	X’5C’ 92	\	X’7C’ 124	|	X’1C’ 28	FS
: ;	X’3B’ 59	;	X’3A’ 58	:	X’3B’ 59	;
” ’	X’27’ 39	’	X’22’ 34	”	X’27’ 39	’
< ,	X’2C’ 44	,	X’3C’ 60	<	X’2C’ 44	,
> .	X’2E’ 46	.	X’3E’ 62	>	X’2E’ 46	.
? /	X’2F’ 47	/	X’3F’ 63	?	X’1F’ 31	US
> <	X’3C’ 60	<	X’3E’ 62	>	X’3C’ 60	<

2.2.6 Terminal Function Keys

The Terminal Function keys in produce codes used by a typical video display terminal. These keys enable an operator to generate any commonly used terminal control character with a single keystroke. (The codes produced by these keys are identical to those generated by the conventional two-key control sequences.)

Note that the SHIFT and CTRL keys have no effect on the codes produced by the Terminal Function keys, except for the *CTRL Space Bar* sequence that generates an ASCII **NUL** character.

Table 13B-5 lists the codes and characters generated by the Terminal Function keys.

Table 13B-5. Terminal Function Keys

Key	*Key Alone*		*SHIFT+Key*		*CTRL+Key*	
Label	*Code*	*Char*	*Code*	*Char*	*Code*	*Char*
BREAK	X'A0' 160		X'A0' 160		X'A0' 160	
SCROLL LOCK	X'9F' 159		X'9F' 159		X'9F' 159	
BACK SPACE	X'08' 8	BS	X'08' 8	BS	X'08' 8	BS
DELETE	X'7F' 127	DEL	X'7F' 127	DEL	X'7F' 127	DEL
RETURN	X'0D' 13	CR	X'0D' 13	CR	X'0D' 13	CR
LINE FEED	X'0A' 10	LF	X'0A' 10	LF	X'0A' 10	LF
ESC	X'1B' 27	ESC	X'1B' 27	ESC	X'1B' 27	ESC
TAB	X'09' 9	HT	X'09' 9	HT	X'09' 9	HT
(none; space bar)	X'20' 32	(space)	X'20' 32	(space)	X'00' 0	NUL

2.2.7 PS 390 Function Keys

The PS 390 Function Keys are used to transmit special 2-byte system sequences. Table 13B-6 shows the the codes for these keys.

Table 13B-6. PS 390 Function Key Codes

Key Label	*Key Alone Code*	*SHIFT+Key Code*	*CTRL+Key Code*
F1	X'1661	X'1641'	X'1601'
F2	X'1662	X'1642'	X'1602'
F3	X'1663'	X'1643'	X'1603'
F4	X'1664'	X'1644'	X'1604'
F5	X'1665'	X'1645'	X'1605'
F6	X'1666'	X'1646'	X'1606'
F7	X'1667'	X'1647'	X'1607'
F8	X'1668'	X'1648'	X'1608'
F9	X'1669'	X'1649'	X'1609'
F10	X'166A'	X'164A'	X'160A'
F11	X'166B'	X'164B'	X'160B'
F12	X'166C'	X'164C'	X'160C'

2.2.8 Numeric/Application Mode Keys

The numeric application mode keys generate special 2-byte PS 390 system sequences similar to those produced by the PS 390 Function keys.

Note that neither SHIFT nor CTRL affects the ENTER key, and that no codes are modified by the CTRL key.

Any code generated by a Numeric/Application Mode key may be duplicated by entering CTRL SHIFT V, followed by the appropriate displayable character or control character.

Table 13B-7 illustrates the codes and characters produced by the Numeric/Application Mode keys.

Table 13B-7. Numeric/Application Mode Key Codes

Key Label	*Key Alone*		*SHIFT+Key*		*CTRL+Key*	
	Code	*Char*	*Code*	*Char*	*Code*	*Char*
0	X'1630'		X'1629'		X'1630'	
1	X'1631'		X'1621'		X'1673'	
2	X'1632'		X'1640'		X'1644'	
3	X'1633'		X'1623'		X'1633'	
4	X'1634'		X'1624'		X'1670'	
5	X'1635'		X'1625'		X'166F'	
6	X'1636'		X'165E'		X'1636'	
7	X'1637'		X'1626'		X'1652'	
8	X'1638'		X'162A'		X'1612'	
9	X'1639'		X'1628'		X'1639'	
.	X'162E'	.	X'163E'	>	X'162E'	.
,	X'162C'	,	X'163C'	<	X'162C'	,
-	X'162D'	(minus) -	X'165F'	(underline) _	X'162D'	-
ENTER	X'160D'	CR	X'160D'	CR	X'160D'	CR

2.2.9 Device Control Keys

The Device Control keys generate two-byte sequences similar to those described in 2.2.7 and 2.2.8. The codes produced by these keys are modified by SHIFT and CTRL as shown in Table 13B-8.

Any code generated by a Device Control key may also be produced by entering *CTRL SHIFT V*, followed by the appropriate displayable character or control character.

Table 13B-8. The Device Control Key Codes

Key	*Key Alone*	*SHIFT+Key*	*CTRL+Key*
Label	*Code*	*Code*	*Code*
1 TERM	X'1631'	X'1621'	X'1673'
2 NRMTST	X'1632'	X'1640'	X'1644'
4 GRAPH	X'1634'	X'1624'	X'1670'
5 SET UP	X'1635'	X'1625'	X'166F'
7 LOCAL	X'1637'	X'1626'	X'1652'
8 CMND	X'1638'	X'162A'	X'1612'
←	X'1677'	X'1657'	X'1617'
→	X'1678'	X'1658'	X'1618'
↑	X'1679'	X'1659'	X'1619'
↓	X'167A'	X'165A'	X'161A'
PF1 HOST	X'A9'	X'A9'	X'1672'
PF2 5080	X'AA'	X'AA'	X'1674'

The Cursor Up key becomes Scroll Up when shifted.
The Cursor Down key becomes Scroll Down when shifted.

3. The Control Dials

The Control Dials consist of an array of 8 shaft encoders arranged in a 2 column x 4 row design, with the number 1 dial being the upper left-hand dial and the number 5 dial being the upper right-hand dial when the Dials are situated in their vertical orientation. The Control Dials report to the Joint Control Processor the number of counts rotated between sampling intervals. The Joint Control Processor may specify the number of counts to be accumulated between sampling intervals and may set a sampling time for all the dials. (Default value for the Dials is 1024 counts per revolution at 4 count increments and 30 samples per second.)

3.1 Control Dial Responses

The Control Dials output relative delta values only. For example; each dial's position is reported in terms of its last sample location. The data format used to report the count is:

Table 13B-9. Control Dial Response Data Format

Byte Number	Description
1	Control V = ´00010110´
2	Byte = ´00000nnn´, Where nnn is a binary number 000 thru 111 (0 thru 7 decimal which specifies the dial.)
3	Most significant byte of a 16-bit signed integer (sign indicates direction).
4	Least significant byte of the 16-bit signed integer (two's complement notation).

3.2 Commands to the Control Dials

The Control Dials must respond to two commands. The first is in the same format as the response message except that the second byte is ´100xxnnn´ and no sign is legal on the 16-bit integer. It specifies the delta value which must be accumulated before the delta count is reported to the host (meaning how many counts between reports).

The second command is formatted as follows and applies a sampling time to all the dials:

Table 13B-10. Control Dial Command Data Format

Byte Number	Description
1	Control V = ´00010110´
2	Control Byte = ´1x1xxxxx´, (x=don't care)
3	Reserved unused byte.
4	Time count in binary, Where x´05´ = 60 samples/second Where x´0A´ = 30 samples/second Where x´1E´ = 10 samples/second

This time indicates how often the Control Dials samples to see if sufficient counts have been accumulated on any dial to respond to the processor.

3.3 Transmission Characteristics

The data sent to and from the Control Dials is asynchronous with each byte containing eight data bits with no parity, one start bit and one stop bit. The data transmission rate of the Control Dials is 9600 baud.

4. The 32-Key Lighted Function Buttons

The Lighted Function Buttons consists of an array of 32 lighted function keys. The Joint Control Processor sends the message to the Function Button Unit that lights the keys which are candidates to be selected to invoke specific program functions. The same message may also turn off some of the lights which are already on. This cues the operator that he may select one of the lighted keys by pressing the key. The Function Buttons Unit then sends a message to the Joint Control Processor which indicates that a specific key has been depressed. The software can then take action(s) based upon the key selection.

4.1 Light Control

The Function Button lights are logically grouped into eight groups of four lights each. The lights of the box are turned on and of respectively by sending a message consisting of one to eight bytes to the unit. The four most significant bits of each byte contains the identification number for a four-light group; the four least significant bits contain a mask which turn on (if the corresponding bit is set) or off (if the bit is clear) the light. This is shown in Table 13B-11 where the Group Number is binary 0000 through 0111 and Light Mask 1's and 0's turn lights on and off.

7	6	5	4	3	2	1	0
Group				Mask			

Table 13B-11. Function Button Light Control Message Byte

The Function Button Light Groups are defined in Table 13B-12.

Table 13B-12. Function Button Light Groups

Group Number	**Description**
b´0000´	Group for lights 1 through 4
b´0001´	Group for lights 5 through 8
b´0010´	Group for lights 9 through 12
b´0011´	Group for lights 13 through 16
b´0100´	Group for lights 17 through 20
b´0101´	Group for lights 21 through 24
b´0110´	Group for lights 25 through 28
b´0111´	Group for lights 29 through 32

Any byte or combination of bytes may be sent in a message, depending on which of the lights must be turned on or turned off. Turning all lights on, turning all lights off or changing the state of at least one byte of each of the eight groups requires an eight-byte message to be sent. Changing the state of one to four lights in a single four-light group requires only a one-byte message to be sent.

4.2 Reporting Selections

The Function Button Unit reports that a key has been pressed by sending a single byte to the Joint Control Processor. The value of the byte is given by adding the hexadecimal value of the key number to the hexadecimal value x´3F´. Thus the first sixteen keys are numbered x´40´ to x´4F´ and the second group of sixteen keys are numbered x´50´ to x´5F´. Only one key depression per message is reported.

4.3 Self Test Command and Report

The Function Buttons Unit has a self-test command and report that is used for diagnostics and optionally for initialization confidence tests. The command is a single byte: x´80´. The response is a four-byte sequence as shown in Table 13B-13.

Table 13B-13. Function Button Self Test Responses

Byte 1	64H, Hardware ID for the Button Box.
Byte 2	xxH, where xx is the firmware revision level. This should begin with 01H.
Byte 3	00H if ROM and RAM test successful and 3EH if ROM or RAM test failed, (RAM and ROM refer to processor chip), or 3DH if key down on Self Test (3E supersedes 3D)
Byte 4	00H on successful test, or xxH, where xx is code of keydown at Self Test.

4.4 Transmission Characteristics

The data sent to and from the Function Buttons Unit is asynchronous data with each byte containing eight data bits without parity plus one start bit and one stop bit. The data transmission rate of the Buttons box is 9600 baud.

5. Data Tablet

There are two data tablets available for use with the PS 390. Both tablets are identical for the PS 300 style and the PS 390 style interactive devices. There is a 6-inch by 6-inch and a 12-inch by 12-inch tablet, each with a four-button puck. Both are alike, except for their active areas, and both provide digitizing and picking functions for the PS 390.

5.1 Operating Modes

Data tablet modes may be controlled externally under program control. The following operating modes are available:

- Point mode

  Pressing a puck button at a given tablet location causes one X,Y coordinate pair (sample) to be transmitted.

- Stream mode

  X,Y coordinate pairs are generated continuously at the selected sampling rate when the puck is near the active area of the tablet.

- Switched stream mode

  Pressing a button on the puck causes X,Y coordinate pairs to be output continuously at the selected sampling rate until the button is released.

Both the mode and the sampling rate may be changed under program control from the PS 390 by sending the data tablet an ASCII character. Table 13B-14 lists the ASCII codes.

Table 13B-14. Binary Data Transmission Codes

Mode	Binary Rate	Uppercase ASCII Character
Stop	-	S
Point	-	P
Switched Stream	2	@
	4	A
	10	B
	20	C
	35	D
	70	E
	141	F
	141	G
Stream	2	H
	4	I
	10	J
	20	K
	35	L
	70	M
	141	N
	141	O

5.1.1 Binary Data Format

The binary formatted RS-232 interface is a five-byte count output. Binary format is shown in Table 13B-15.

Table 13B-15. Data Tablet Binary Format

Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	P	1	F3	F2	F1	F0	0	0
2	P	0	X5	X4	X3	X2	X1	X0
3	P	0	X11	X10	X9	X8	X7	X6
4	P	0	Y5	Y4	Y3	Y2	Y1	Y0
5	P	0	Y11	Y10	Y9	Y8	Y7	Y6

6. The Optical Mouse

The optical mouse transforms position information into a digital form acceptable to the PS 390. The optical mouse uses a three-button mouse unit in conjunction with a reflective pad to provide X- and Y-axis position information.

The mouse uses LEDs reflecting off the pad to provide directional information to the control logic in the mouse. This movement is then translated into relative X and Y movement information. The data is transmitted serially to the PS 390 through the peripheral multiplexer.

NOTE

The optical mouse pad must be oriented horizontally to the user for proper mouse operation. Furthermore, the mouse cord (tail) should lead away from the user.

6.1 Protocol

The mouse protocol is 9600 baud asynchronous serial with one start bit, one stop bit, and eight data bits. The least significant data bit is transmitted first. Blocks of five bytes are sent whenever there is a change of mouse state (switches or position) since the last transmission. The protocol is as follows:

1. Byte 1: Bits 3 through 7 represent the sync for the start of the data block with bit 7 = 1 and bits 3–6 = 0. Bits 0 through 2 define switch status (0 switches the depressed state). With the mouse oriented so that the cord is facing away from the user, the right switch status is indicated by bit 0, the middle switch status by bit 1, and the left switch status is indicated by bit 2.

2. Byte 2: Bits 0 through 7 represent the incremental change in the X-direction since the last complete report up to the time Byte 1 starts transmission. The data is in the two's complement form and has a value limit of +/- 127. With the mouse cord facing away from the user, moving the mouse to the right produces positive X values and moving the mouse to the left produces negative X values.

3. Byte 3: Bits 0 through 7 represent the incremental change in the Y-direction since the last complete report up to the time Byte 1 starts transmission. The data is in the two's complement form and has a value limit of +/- 127. Moving the mouse towards its mouse cord produces positive X values and moving the mouse away from its cord produces negative Y values.

4. Byte 4: Bits 0 through 7 follow the same format as Byte 2 and represent the data acquired since the beginning of Byte 1 transmission.

5. Byte 5: Bits 0 through 7 follow the same format as Byte 3 and represent the data acquired since the beginning of Byte 4 transmission.

Table 13B-16. Mouse Bit Protocol

	MSB							LSB
Bit No.	7	6	5	4	3	2	1	0
Byte 1	1	0	0	0	0	L	M	R
Byte 2	X7	X6	X5	X4	X3	X2	X1	X0
Byte 3	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
Byte 4	X7	X6	X5	X4	X3	X2	X1	X0
Byte 5	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0

RM14. GSR INTERNALS

CONTENTS

Section RM14
GSR Internals

This section describes the data formats expected by PS 390 command interpreter (CI) and other intrinsic functions. It provides you with the necessary information to write your own GSRs.

NOTE

Information in this section is based on information in other sections of this guide. Where helpful, information will be duplicated here for clarity. Otherwise, references will be given to other sections as necessary.

The first section discusses formats for the different data types. It is important to note that data is received a byte at a time by the JCP. Therefore, where there is a most significant bit (MSB) – least significant bit (LSB) specified, the MSB must be sent first. Error formats and reset commands are also discussed. Reset commands should be sent anytime there is an error detected during the sending of other commands. Reset causes the CI to reset and begin command interpretation again. This section also describes the intrinsic functions that internally receive, pass, and route data.

The second section provides the data formats for each of the commands that can be sent to the CI. The format shows the data type followed by the data that should be sent. The data is expressed as a number, a Boolean value, an identifier or an expression.

The final section provides the 6-bit binary encoding method that can be used by the PS 390 for hosts that can't send binary data. This format uses 2D or 3D vector normalized data as an example.

1. Data Types

This section gives the formats for different data types. Some of the data types that can be passed internally in the PS 390 are defined below:

```
  {0}  Qreset,        {dataless: reset a function instance}
  {1}  Qprompt,       {dataless: flush the CI pipeline}
  {2}  QBoolean,      {normal carrier of Boolean values}
  {3}  Qinteger,      {normal carrier of integer values}
  {4}  Qreal,         {normal carrier of floating point values}
  {5}  Qstring,       {original carrier of byte strings, not used}
  {6}  Qpacket,       {carrier of byte strings}
  {7}  Qmorepacket,   {continuation Qpacket carrier of byte strings}
  {8}  Qmove2,        {2D vector including P bit}
  {9}  Qdraw2,        {2D vector including L bit}
 {10}  Qvec2,         {2D vector with no P/L bit (normal vector)}
 {11}  Qmove3,        {3D vector including P bit}
 {12}  Qdraw3,        {3D vector including L bit}
 {13}  Qvec3,         {3D vector with no P/L bit (normal vector)}
 {14}  Qmove4,        {4D vector including P bit}
 {15}  Qdraw4,        {4D vector including L bit}
 {16}  Qvec4,         {4D vector with no P/L bit (normal vector)}
 {17}  Qmat2,         {2x2 matrix}
 {18}  Qmat3,         {3x3 matrix}
 {19}  Qmat4,         {4x4 matrix}
 {20}  Qbindata       {definition of binary data (data part of vector
                      list)}
 {21}  Qusertype      {type that user may use to define own message}
           .
           .
           .
     );
```

Qdtype is padded with 260 miscellaneous elements to ensure that a 16-bit field is allocated by the Pascal compiler rather than the 8-bit field that would be allocated otherwise.

The Qdtype is used to specify the different types of Qdata message blocks available in the PS 390 runtime system. Qdata blocks are the primary vehicle for communication in the PS 390. When a Qdata message is input to a function, it checks to see if it is a valid message type (Qdtype). When a message is output by a function, it carves a Qdata message of the appropriate type and outputs it.

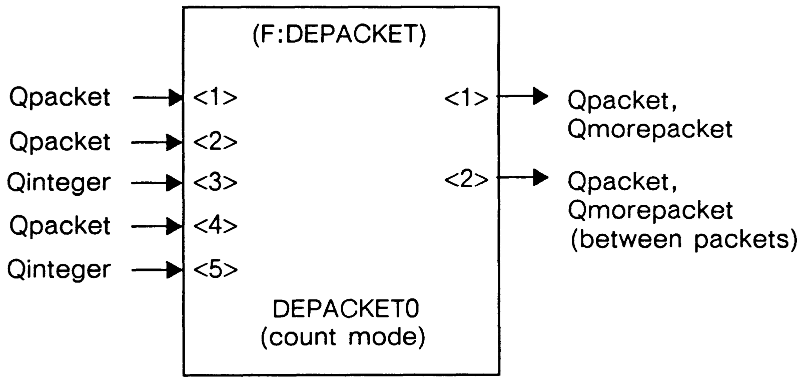
The CI expects tokens that consist of a size, a data type, and a value. Once given, the type of command is implicit in the type of the token, such as "Qsetcontrast" for "Set Contrast." The CI accepts tokens until it has enough to carry out a command.

1.1. Routing Functions

Data is sent from the host to the PS 390 as a stream of bytes. The bytes contain information that tells the PS 390 intrinsic functions the nature of the message and where it is to be sent internally. The following is a list of the data transfer modes used in host/PS 390 communication and a brief description of the intrinsic functions that accept, examine, and route data internally in the PS 390.

F:DEPACKET

An intrinsic user function, F:DEPACKET, accepts data (input to the PS 390 from the host) from receiving functions (B1$, etc.). F:DEPACKET converts a stream of bytes from the host into a stream of Qpacket/Qmorepacket. A Qpacket is a block of character data that can be sent from one PS 390 function to another. When data comes from the host through the F:DEPACKET function, it contains a byte for routing control. A Qmorepacket is a Qpacket that when coming from the host through F:DEPACKET, has no routing byte. A Qmorepacket has the same destination as the previous Qpacket.



In count mode, F:DEPACKET assumes that a packet is defined as:

<SOP>	count bytes	packet contents

where <SOP> represents the Start of Packet (SOP) character that is by default the the ASCII ACK character, decimal character code 06 (^F).

The definition of SOP (one character) is taken from a single character Qpacket on input <2>.

The message count is defined by **n** bytes (**n** defined by the Qinteger on input <3>). Each count byte is offset from the base character (the base character is taken from a single character Qpacket on input <4>). After the base character is subtracted, each count byte becomes a digit of the message count whose radix is defined by the Qinteger on input <5>.

Output <1> outputs Qpackets and Qmorepackets of count mode messages. Output <2> outputs Qpackets and Qmorepackets of any messages which are not in count mode.

The <SOP> byte and the count bytes are removed from the start of the packet before the packet is sent to F:CIROUTE, which does the actual routing.

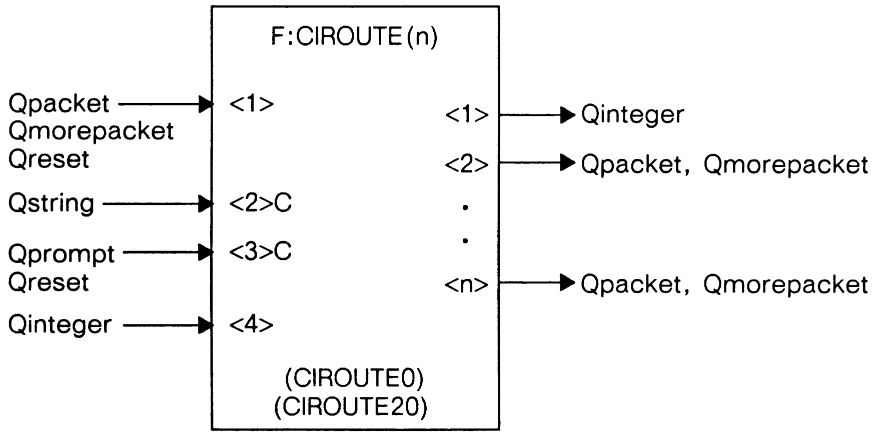
F:CIROUTE(n)

Once data has passed through an instance of F:DEPACKET, the next function to receive it is F:CIROUTE(n). F:CIROUTE(n) has two instances, one for count mode and one for escape mode. Count and escape mode are functionally similar; therefore, only the count mode instance, CIROUTE0, will be described. CIROUTE0 examines the first character of the Qpackets it receives (the character following the count bytes in count mode, or the character following the <FS> character in escape mode) to determine where the packet message is to be sent. These characters are routing bytes, and are used to select the appropriate channel for data in the PS 390.

Data channels include lines to:

- Terminal emulator
- PS 390 CI (through F:READSTREAM for binary packets)
- Disk writing function
- Other intrinsic functions

A base character, defined on Input <2> of CIROUTE0, is subtracted from this routing byte before it is used to select the output channel. The base character defaults to the character zero ("0").



F:CIROUTE demultiplexes a stream of Qpackets/Qmorepackets from input <1> to one of the n output channels. The first byte of an incoming Qpacket is assumed to be the multiplexing byte, equal to the base character (from input <2>) + K, where K is the channel number. If K >(n–3) or K < 0, there is no channel for this output and a pair of messages are sent on outputs <1> and <2>. These can be used to allow for later remultiplexing or further demultiplexing. An integer giving the indicated output port is sent on output <1> and the message for which there was no defined output is sent on output <2>. Whether or not K is within the limits implied by the number of outputs of F:CIROUTE, the multiplexing byte is removed from the start of the packet.

F:CIROUTE passes incoming Qmorepackets out the current channel (as defined by the last Qpacket). Initially, after a Qreset is received, the current channel is –1.

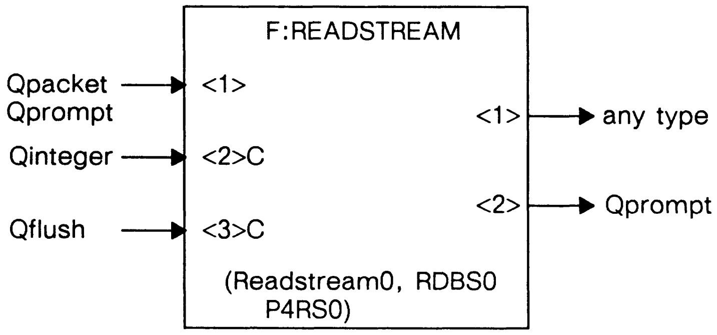
When instancing this function, a parameter is required to specify the number of outputs.

F:CIROUTE(n) is a special version of F:DEMUX(n). It assumes that it is driving parallel, asynchronous paths to a common destination, the CI. F:CIROUTE(n) synchronizes the paths by sending a Qprompt at the end of a channel, then waiting for it to come back around before switching to the next channel. This assumes that the CI can strip Qprompts and send them back. Input <4> gives the maximum channel number, m, for which path flushing is desired. F:CIROUTE(n) flushes channels 0<=K <= m with Qprompts.

The definitions for the inputs and outputs for F:CIROUTE(n), and routing bytes used by F:CIROUTE(n) are described in Section *RM2, Intrinsic Functions.*

F:READSTREAM

Binary packet data sent from F:CIROUTE(n) to the CI is sent through F:READSTREAM. This is the same path the GSRs take.

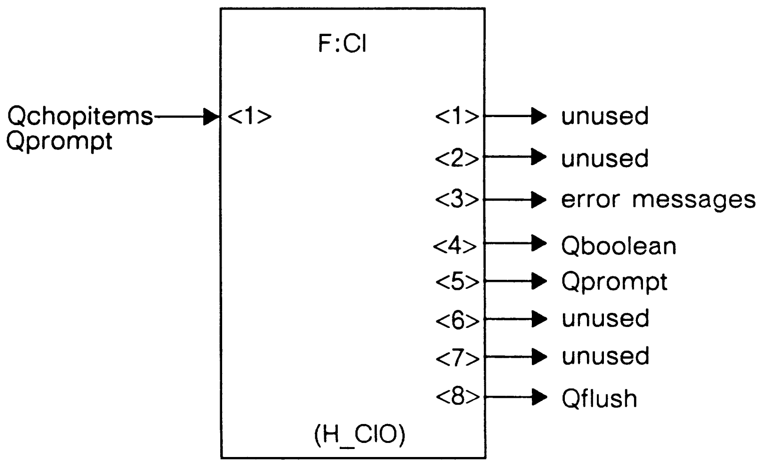


This function converts an 8-bit stream into arbitrary messages. It takes two bytes as the count of information (including message type) and creates a message of that size with the bytes of information that follow it. The message format on input <1> is:

2 bytes	2 bytes	
length	message type	rest of message body

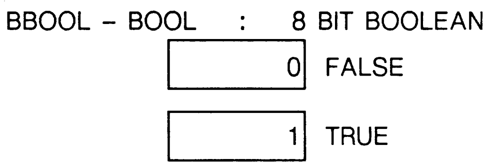
F:CI

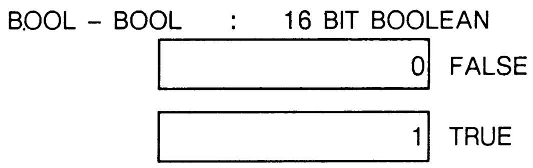
The CI accepts messages from the GSRs through an instance of F:READSTREAM.



This function interprets commands, creating display structures and function networks. It receives input either from a chop/parse function or a READSTREAM function (if using the GSRs).

1.2. Data Formats for Data Types





INT8 - BYTE : 8 BIT INTEGER

INT16 - WORD : 16 BIT INTEGER

MSB	LSB

INT32 - LWORD : 32 BIT INTEGER

MSB	LSB
MSB	LSB

PSREAL - REAL32 : 64 BIT REAL

S| MSB	LSB	EXPONENT
S| MSB	LSB	MS 16 BITS OF FRACTION
MSB	LSB	LS 16 BITS OF FRACTION
	0	PADDING BYTES

NOTE

All exponents are signed integers in the range of +/- 1024. All fractions have their sign bits in the most significant bit of the fraction.

ID - NAME, SIZE

	CHARACTER NAME(1)
:	
	CHARACTER NAME(SIZE)

STRING - NAME, SIZE

	CHARACTER STR(1)
:	
	CHARACTER STR(SIZE)

VECNO - V, POSLIN, DIM, COUNT

COUNT OF

2D VECTOR - MOVE

S| MSB	LSB	X NORMALIZED FRACTION
MSB	LSB	Y NORMALIZED FRACTION
EXP	INTENS|0	EXPONENT/INTENSITY - MOVE

2D VECTOR - DRAW

MSB	LSB	X NORMALIZED FRACTION
MSB	LSB	Y NORMALIZED FRACTION
EXP	INTENS|1	EXPONENT/INTENSITY - DRAW

or

3D VECTOR - MOVE

MSB	LSB	X NORMALIZED FRACTION
MSB	LSB	Y NORMALIZED FRACTION
MSB	LSB	Z NORMALIZED FRACTION
EXP	INTENS|0	EXPONENT/INTENSITY - MOVE

3D VECTOR - DRAW

MSB	LSB	X NORMALIZED FRACTION
MSB	LSB	Y NORMALIZED FRACTION
MSB	LSB	Z NORMALIZED FRACTION
EXP	INTENS|1	EXPONENT/INTENSITY - DRAW

VBLNO - POSLIN, DIM, COUNT

EXP	INTENS	EXPONENT/INTENSITY

FOLLOWED BY COUNT OF

2D VECTOR - MOVE

MSB	LSB	X NORMALIZED FRACTION
MSB	LSB |0	Y NORMALIZED FRACTION - MOVE

2D VECTOR - DRAW

MSB	LSB	X NORMALIZED FRACTION
MSB	LSB |1	Y NORMALIZED FRACTION - DRAW

or

3D VECTOR - MOVE

MSB	LSB	X NORMALIZED FRACTION
MSB	LSB	Y NORMALIZED FRACTION
MSB	LSB |0	Z NORMALIZED FRACTION - MOVE

3D VECTOR - DRAW

MSB	LSB	X NORMALIZED FRACTION
MSB	LSB	Y NORMALIZED FRACTION
MSB	LSB |1	Z NORMALIZED FRACTION - DRAW

1.3. Error Formatting

This format is used to reset the CI after an error.

```
ERROR  - ERRCOD

INT16 - 2
INT16 - QERRFL=143
```

2. Command Interpreter Data Format

This section provides the data formats for most of the commands that can be sent to the PS 390 CI.

The format shows the data type, followed by the data that should be sent. The data is expressed as a number, a Boolean value, an identifier, or an expression. If an identifier begins with the letter Q, it is a subcommand type and the value of the subcommand to be used is shown after the equal sign (=). If the identifier is SIZE it refers to the size or length of the string or ID about to be transferred. All other identifiers are user supplied variables.

2.1. Data Format Analysis

To help understand how PS 390 commands are built from subcommands, the structure of some commands is analyzed below. Note that each Qdata (subcommand) described has the same substructure, as follows:

- Number of bytes in the Qdata
- The tag identifying the particular Qdata
- The data, if any

Data that may vary in size, such as character strings, is structured such that the CI can deal with it correctly.

The following describes how the pieces of information are incorporated into the data sent by the GSRs to the CI.

2.1.1. Example — Character Rotate Command

The command:

```
Handle := CHARACTER ROTATE angle APPLIED TO Apply;
```

has three parts, as follows:

```
1. Handle :=
2. CHARACTER ROTATE angle
3. APPLIED TO Apply
```

This Qdata describes the **Handle :=** part of the command.

```
INT16 - SIZE+8       { A Qdata always starts with a byte count }
INT16 - QLABEL=44    { This particular Qdata is a QLabel }
INT16 - SIZE         { The number of bytes in the name "Handle" }
INT16 - 1            { always starts at the first byte }
ID    - HANDLE,SIZE  { A array of bytes containing the string "Handle" }
INT16 - 0            { always a 0 }
```

This Qdata describes the **CHARACTER ROTATE angle** part of the command.

```
INT16 - 10           { This particular Qdata is 10 bytes long }
INT16 - QROTTXT=77   { And is a character rotate command }
PSREAL- ANGLE        { with a rotation angle of "ANGLE" }
```

This QData describes the **APPLIED TO Apply** part of the command.

```
INT16 - SIZE+8       { The byte count of the qdata}
INT16 - QNAME=45     { This particular Qdata is a QNAME }
INT16 - SIZE         { the number of bytes in the name "APPLY" }
INT16 - 1            { starts a byte position 1 }
ID    - APPLY,SIZE   { the array of bytes containing the string "APPLY"}
INT16 - 0            { always a 0 }
```

Contrast this command with others of the same form such as:

```
Handle := TRANSLATE X,Y,Z APPLIED TO Apply;.
```

2.1.2. Example — Connect Command

The command:

```
CONNECT SOURCE<OUT>:<INP>DEST;
```

has several parts, as follows:

1. The command verb **CONNECT**
2. The source of the connection **SOURCE**
3. The particular output of the source **<OUT>**
4. The input number of the connection destination **<INP>**
5. The destination of the connection **DEST**

The Qdata sent by the GSR's for this command reflects this structure.

This Qdata tells the CI to look up the name **SOURCE**. Note the similarity to QNAME and QLABEL in the examples.

```
INT16 - SIZE+8
INT16 - QALOOK=100
INT16 - SIZE
INT16 - 1
ID    - SOURCE,SIZE
INT16 - 0
```

This Qdata identifies the output number of **SOURCE**.

```
INT16 - 6
INT16 - QFNOUT=144
INT32 - OUT
```

This Qdata is another **QALOOK,** instructing the CI to look up the name **DEST**.

```
INT16 - SIZE+8
INT16 - QALOOK=100
INT16 - SIZE
INT16 - 1
ID    - DEST,SIZE
INT16 - 0
```

This Qdata identifies the input number of **DEST** to connect to.

```
INT16 - 6
INT16 - QINPIN=145
INT32 - INP
```

This Qdata identifies the command as a **CONNECT** command.

```
INT16 - 2
INT16 - QCON=138
```

Contrast this command with the DISCONNECT and SEND commands.

2.2. Data Formats

```
HANDLE := ATTRIBUTES [COLOR hue[,sat[,intens]]]
     [DIFFUSE diffus]
     [SPECULAR specul];

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 44
      INT16 - QATTR=357
      PSREAL- HUE
      PSREAL- SAT
      PSREAL- INTENS
      PSREAL- 0.
      PSREAL- DIFFUS
      INT16 - SPECUL


HANDLE := ATTRIBUTES [COLOR hue[,sat[,intens]]]
     [DIFFUSE diffus]
     [SPECULAR specul]
      AND   [COLOR hue2[,sat2[,inten2]]]
     [DIFFUSE diffu2]
     [SPECULAR specu2];

      INT16 - SIZE+8
      INT16 - QLABEL=44
```

```
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 86
        INT16 - QOATTR=358
        PSREAL- HUE
        PSREAL- SAT
        PSREAL- INTENS
        PSREAL- 0.
        PSREAL- DIFFUS
        INT16 - SPECUL
        PSREAL- HUE2
        PSREAL- SAT2
        PSREAL- INTEN2
        PSREAL- 0.
        PSREAL- DIFFU2
        INT16 - SPECU2


BEGIN

        INT16 - 2
        INT16 - QBEGIN=105

HANDLE := BEGIN_STRUCTURE

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 2
        INT16 - QBEGOB=103

HANDLE := BSPLINE
             ORDER = ORDER
             OPEN/CLOSED
             NONPERIODIC/PERIODIC
             N = NVERT
             VERTICES   = X(1),    Y(1),    ( Z(1)   )
                          X(2),    Y(2),    ( Z(2)   )
                           :        :          :
                          X(N),    Y(N),    ( Z(N)   )
             KNOTS = KNOTS (1), ... KNOTS (NKNOTS)
             CHORDS = CHORDS;

        INT16 - SIZE+8
```

```
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 14
      INT16 - QSTRTC=152
      INT8  - 1
      BBOOL - .FALSE.
      INT8  - ORDER
      BBOOL - .FALSE.
      INT8  - DIMEN
      BBOOL - (.NOT.OPNCLS)
      BBOOL - (.NOT.NONPER)
      BBOOL - .FALSE.
      INT32 - NVERT

        REPEAT NVERT TIMES
      INT16 - 34
      INT16 - QCRVEC=296
      PSREAL- V (1,1)
      PSREAL- V (2,1)
      PSREAL- V (3,1)
      PSREAL- V (4,1)

      (OPTIONAL)
        REPEAT NKNOTS TIMES
      INT16 - 10
      INT16 - QKNOT=295
      PSREAL- KNOTS (I)

      INT16 - 14
      INT16 - QENDCV=153
      INT32 - CHORDS
      PSREAL- 0

HANDLE := CHARACTER ROTATE ANGLE (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 10
      INT16 - QROTTX=77
      PSREAL- ANGLE
      INT16 - SIZE+8
      INT16 - QNAME=45
```

```
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0

HANDLE := CHARACTERS TRANX,TRANY,TRANZ
          STEP STEPX,STEPY 'CHARS';

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 22
        INT16 - QTXTLB=159
        PSREAL- STEPX
        PSREAL- STEPY
        INT32 - 0
        INT16 - SIZE + 6
        INT16 - QDTSTR=305
        INT16 - SIZE
        INT16 - 1
        STRING- CHARS, SIZE
        INT16 - 26
        INT16 - Q3DPCH=306
        PSREAL- TRANX
        PSREAL- TRANY
        PSREAL- TRANZ
        INT16 - 2
        INT16 - QENDCH=304


HANDLE := CHARACTER SCALE SCALEX, SCALEY
          (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 18
        INT16 - QTXTSC=166
        PSREAL- SCALEX
        PSREAL- SCALEY
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
CONN SOURCE <OUT>:<INP> DEST;

        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - SOURCE, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QFNOUT=144
        INT32 - OUT
        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - DEST, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QINPIN=145
        INT32 - INP
        INT16 - 2
        INT16 - QCON=138
```

```
HANDLE := COPY CPYFRM (START=) START (,) (COUNT=) COUNT;

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - CPYFRM, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QCOPY=123
        INT16 - START
        INT16 - COUNT
```

```
HANDLE1 := PATTERN i (i) [AROUND_CORNERS] [MATCHiNOMATCH]
                  LENGTH l;

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE1, SIZE
      INT16 - 0
      INT16 - 46
      INT16 - QPATRN=149
      BBOOL - .NOT. CONTIN
      BBOOL - MATCH
      PSREAL- LENGTH
      INT8  - SEGS ( 0<SEGS<=32 )
      INT8  - 0
      INT8  - PATTRN (1 TO SEGS)

        IF SEGS < 32 REPEAT TO EQUAL 32 INT8 VALUES
      INT8  - 0
      INT16 - 2
      INT16 - QENDCH=304


DELETE HANDLE;

      INT16 - 2
      INT16 - QDELET=237
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0


HANDLE := DECREMENT LEVEL_OF_DETAIL
          (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 2
      INT16 - QDECLV=134
      INT16 - SIZE+8
```

```
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
DEL HANDLE*;    (WILD CARD DELETE COMMAND)

        INT16 - 2
        INT16 - QDELW=57
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
```

```
DISCONNECT SOURCE <OUT>:<INP> DEST;

        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - SOURCE, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QFNOUT=144
        INT32 - OUT
        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - DEST, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QINPIN=145
        INT32 - INP
        INT16 - 2
        INT16 - QDISCN=139
```

```
DISCONN SOURCE:ALL;

        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
```

```
      INT16 - 1
      ID    - SOURCE, SIZE
      INT16 - 0
      INT16 - 2
      INT16 - QALLDS=219


DISCONNECT SOURCE <OUT>:ALL;

      INT16 - SIZE+8
      INT16 - QALOOK=100
      INT16 - SIZE
      INT16 - 1
      ID    - SOURCE, SIZE
      INT16 - 0
      INT16 - 6
      INT16 - QFNOUT=144
      INT32 - OUT
      INT16 - 2
      INT16 - QALLDS=219


DISPLAY HANDLE;

      INT16 - 2
      INT16 - QDSPOB=118
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0


END;

      INT16 - 2
      INT16 - QEND=106


END OPTIMIZE;

      INT16 - 4
      INT16 - QOPTIM=162
      BOOL  - .FALSE.
```

```
END_STRUCTURE;

      INT16 - 2
      INT16 - QENDOB=104



ERASE PATTERN FROM HANDLE;

      INT16 - SIZE+8
      INT16 - QERAPA=332
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0



HANDLE := EYE BACK DISTB
        LEFT/RIGHT DISTLR
        UP/DOWN DISTUD
        FROM SCREEN AREA WIDTH WIDE
        FRONT BOUNDARY = FRONT
        BACK BOUNDARY  = BACK
        (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 50
      INT16 - QEYE=155
      PSREAL- DISTLR
      PSREAL- DISTUD
      PSREAL- -DISTB
      PSREAL- WIDE
      PSREAL- FRONT
      PSREAL- BACK
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0
```

```
HANDLE := F:FNNAME;

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - SIZE + 12
      INT16 - QFLOOK=99
      INT16 - 0
      INT32 - 0
      INT16 - SIZE
      ID    - FNNAME, SIZE
      INT16 - 0


HANDLE := F:FNNAME (INOUTS);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - SIZE + 12
      INT16 - QPARFN=267
      INT16 - INOUTS
      INT32 - 0
      INT16 - SIZE
      ID    - FNNAME, SIZE
      INT16 - 0


FOLLOW HANDLE WITH TRANSFORMATION-OR-ATTRIBUTE COMMAND;

      INT16 - 2
      INT16 - QFOLLO=115

      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
```

```
HANDLE := CHARACTER FONT FONTNM (APPLIED TO APPLY);

       INT16 - SIZE+8
       INT16 - QLABEL=44
       INT16 - SIZE
       INT16 - 1
       ID    - HANDLE, SIZE
       INT16 - 0
       INT16 - 2
       INT16 - QUFONT=131
       INT16 - SIZE+8
       INT16 - QNAME=45
       INT16 - SIZE
       INT16 - 1
       ID    - FONTNM, SIZE
       INT16 - 0
       INT16 - SIZE+8
       INT16 - QNAME=45
       INT16 - SIZE
       INT16 - 1
       ID    - APPLY, SIZE
       INT16 - 0

FORGET HANDLE;

       INT16 - 2
       INT16 - QFORG=113
       INT16 - SIZE+8
       INT16 - QNAME=45
       INT16 - SIZE
       INT16 - 1
       ID    - HANDLE, SIZE
       INT16 - 0


HANDLE := FIELD_OF_VIEW ANGLE
         FRONT BOUNDARY = FRONT
         BACK BOUNDARY  = BACK
         (APPLIED TO APPLY);

       INT16 - SIZE+8
       INT16 - QLABEL=44
       INT16 - SIZE
       INT16 - 1
       ID    - HANDLE, SIZE
       INT16 - 0
       INT16 - 24
       INT16 - QFOV=156
```

```
          PSREAL- ANGLE
          PSREAL- FRONT
          PSREAL- BACK
          INT16 - SIZE+8
          INT16 - QNAME=45
          INT16 - SIZE
          INT16 - 1
          ID    - APPLY, SIZE
          INT16 - 0


HANDLE := IF CONDITIONAL_BIT BITNUM IS ONOFF
          (APPLIED TO APPLY);

          INT16 - SIZE+8
          INT16 - QLABEL=44
          INT16 - SIZE
          INT16 - 1
          ID    - HANDLE, SIZE
          INT16 - 0
          INT16 - 8
          INT16 - QCOND=174
          BOOL  - ONOFF
          INT32 - BITNUM
          INT16 - SIZE+8
          INT16 - QNAME=45
          INT16 - SIZE
          INT16 - 1
          ID    - APPLY, SIZE
          INT16 - 0


HANDLE := IF LEVEL_OF_DETAIL COMP LEVEL
         (APPLIED TO APPLY);

          INT16 - SIZE+8
          INT16 - QLABEL=44
          INT16 - SIZE
          INT16 - 1
          ID    - HANDLE, SIZE
          INT16 - 0
          INT16 - 8
          INT16 - QCOND=174
          INT16 - (COMP + 2) * 256
          INT32 - LEVEL
          INT16 - SIZE+8
          INT16 - QNAME=45
          INT16 - SIZE
```

```
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0


HANDLE := IF PHASE ONOFF (THEN APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 8
        INT16 - QCOND=174
        BOOL  - ONOFF
        INT32 - 15
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0


HANDLE := ILLUMINATION x,y,z,
        [COLOR hue[,sat[,intens]]]
        [AMBIENT ambien];

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 66
        INT16 - QLGHTS=355
        PSREAL- X
        PSREAL- Y
        PSREAL- Z
        PSREAL- 1.
        PSREAL- HUE
        PSREAL- SAT
        PSREAL- INTENS
        PSREAL- AMBIEN
```

```
INCLUDE HANDLE1 IN HANDLE2;

      INT16 - 2
      INT16 - QSETAD=125
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE1, SIZE
      INT16 - 0
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE2, SIZE
      INT16 - 0


PINIT:  INITIALIZE

      INT16 - 2
      INT16 - QINITN=121
      INT16 - 2
      INT16 - QINITD=122
      INT16 - 2
      INT16 - QINITL=293


INITIALIZE CONNECTIONS;

      INT16 - 2
      INT16 - QINITC=218


INITIALIZE DISPLAYS;

      INT16 - 2
      INT16 - QINITD=122


INITIALIZE HANDLES;

      INT16 - 2
      INT16 - QINITN=121
```

```
HANDLE := INCREMENT LEVEL_OF_DETAIL
               (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 2
      INT16 - QINCLV=133
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0


HANDLE1 := INSTANCE (OF HANDLE2);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE1, SIZE
      INT16 - 0
      INT16 - 2
      INT16 - QUSE=120
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE2, SIZE
      INT16 - 0
      INT16 - 2
      INT16 - QENDLS=107


HANDLE := LABEL X, Y, Z, 'STRING'
              :  :  :      :
              X, Y, Z, 'STRING';

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - LABBLK, SIZE
```

```
        INT16 - 0
        INT16 - 26
        INT16 - QDELTA=308
        PSREAL- STEPX
        PSREAL- STEPY
        PSREAL- 0

         THE NEXT 10 LINES FOR EACH LABEL
        INT16 - SIZE + 6
        INT16 - QDSTR=305
        INT16 - SIZE
        INT16 - 1
        STRING- LABEL, SIZE
        INT16 - 0
        INT16 - 26
        INT16 - Q3DPCH=306
        PSREAL- X
        PSREAL- Y
        PSREAL- Z
        INT16 - 2
        INT16 - QENDCH=304


  HANDLE := LOOK AT AT FROM FROM UP UP (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 90
        INT16 - QLKAT=158
        PSREAL- FROM(1)
        PSREAL- FROM(2)
        PSREAL- FROM(3)
        PSREAL- 0
        PSREAL- AT(1)
        PSREAL- AT(2)
        PSREAL- AT(3)
        PSREAL- 0
        PSREAL- UP(1)
        PSREAL- UP(2)
        PSREAL- UP(3)
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0


HANDLE := MATRIX_2X2 (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 50
        INT16 - QMAT2=17
        PSREAL- MATRIX (1,1)
        PSREAL- MATRIX (1,2)
        PSREAL- 0
        PSREAL- 0
        PSREAL- MATRIX (2,1)
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0


HANDLE := MATRIX_3X3 (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 90
        INT16 - QMAT3=18
        PSREAL- MATRIX (1,1)
        PSREAL- MATRIX (1,2)
        PSREAL- MATRIX (1,3)
        PSREAL- 0
        PSREAL- MATRIX (2,1)
        PSREAL- MATRIX (2,2)
        PSREAL- MATRIX (2,3)
        PSREAL- 0
        PSREAL- MATRIX (3,1)
        PSREAL- MATRIX (3,2)
        PSREAL- MATRIX (3,3)
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
HANDLE := MATRIX_4X3 MAT VEC (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 122
        INT16 - QMATRN=206
        PSREAL- MAT(1,1)
        PSREAL- MAT(1,2)
        PSREAL- MAT(1,3)
        PSREAL- 0
        PSREAL- MAT(2,1)
        PSREAL- MAT(2,2)
        PSREAL- MAT(2,3)
        PSREAL- 0
        PSREAL- MAT(3,1)
        PSREAL- MAT(3,2)
        PSREAL- MAT(3,3)
        PSREAL- 0
        PSREAL- VEC(1)
        PSREAL- VEC(2)
        PSREAL- VEC(3)
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
HANDLE := MATRIX_4X4 (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID     - HANDLE, SIZE
        INT16 - 0
        INT16 - 130
        INT16 - QMAT4=19
        PSREAL- MATRIX (1,1)
        PSREAL- MATRIX (1,2)
        PSREAL- MATRIX (1,3)
        PSREAL- MATRIX (1,4)
        PSREAL- MATRIX (2,1)
        PSREAL- MATRIX (2,2)
        PSREAL- MATRIX (2,3)
        PSREAL- MATRIX (2,4)
        PSREAL- MATRIX (3,1)
        PSREAL- MATRIX (3,2)
        PSREAL- MATRIX (3,3)
        PSREAL- MATRIX (3,4)
        PSREAL- MATRIX (4,1)
        PSREAL- MATRIX (4,2)
        PSREAL- MATRIX (4,3)
        PSREAL- MATRIX (4,4)
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID     - APPLY, SIZE
        INT16 - 0


HANDLE := NIL;

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID     - HANDLE, SIZE
        INT16 - 0
        INT16 - 2
        INT16 - QMKNIL=236


OPTIMIZE STRUCTURE;

        INT16 - 4
        INT16 - QOPTIM=162
        BOOL   - .TRUE.
```

```
PATTERN HANDLE WITH PATNAM;

      INT16 - SIZE+8
      INT16 - QNAMPA=316
      INT16 - SIZE
      INT16 - 1
      ID    - PATNAM, SIZE
      INT16 - 0
      INT16 - SIZE+8
      INT16 - QAPPPA=333
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0


HANDLE :=  [WITH [ATTRIBUTES attr] [OUTLINE r]]
         POLYGON [Coplanar] ( [S] x,y,z [N x,y,z] ) )
          :         :                :
             [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [Coplanar] ( [S] x,y,z [N x,y,z] ) )];

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - SIZE+8
      INT16 - QWTATT=349
      INT16 - SIZE
      INT16 - 1
      ID    - ATTR, SIZE
      INT16 - 0
      INT16 - NVERTS * 8 + 4
      INT16 - QNORML=354
      INT16 - NVERTS
      VECNO - NORMS, VEDGES, DIMEN, NVERTS
      INT16 - NVERTS * 8 + 4
      INT16 - QPOLYG=318 OR QCOPOL=319
      INT16 - NVERTS
      VECNO - VERTS, VEDGES, DIMEN, NVERTS
      INT16 - 2
      INT16 - QEPOLY=320
```

```
HANDLE := POLYNOMIAL
          ORDER = ORDER
          (DIMEN IMPLIED IN SYNTAX)
          COEFFICIENTS = X(I),    Y(I),    Z(I)
                         X(I-1), Y(I-1), Z(I-1)
                           :       :       :
                         X(0),    Y(0),    Z(0)
          CHORDS = CHORDS;

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 14
      INT16 - QSTRTC=152
      INT8  - 2
      BBOOL - .FALSE.
      INT8  - ORDER
      BBOOL - .FALSE.
      INT8  - DIMEN
      BBOOL - (.TRUE.)
      BBOOL - (.TRUE.)
      BBOOL - .FALSE.
      INT32 - ORDER+1

        REPEAT ORDER+1 TIMES
      INT16 - 34
      INT16 - QCRVEC=296
      PSREAL- V (1,1)
      PSREAL- V (2,1)
      PSREAL- V (3,1)
      PSREAL- V (4,1)
      INT16 - 14
      INT16 - QENDCV=153
      INT32 - CHORDS
      PSREAL- 0


PREFIX HANDLE WITH TRANSFORMATION-OR-ATTRIBUTE COMMAND;

      INT16 - 2
      INT16 - QPREFX=114
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0


HANDLE := RAWBLOCK NUMBYTE (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 6
      INT16 - QRAWBL=350
      INT32 - NUMBYTE
       INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0


HANDLE := RATIONAL BSPLINE
        ORDER = ORDER
        OPEN/CLOSED
        NONPERIODIC/PERIODIC
        N = NVERT
        VERTICES    = X(1), Y(1), ( Z(1), ) W(1)
                      X(2), Y(2), ( Z(2), ) W(2)
                        :     :       :
                      X(N), Y(N), ( Z(N), ) W(N)
        KNOTS = KNOTS (1), ... KNOTS (NKNOTS)
        CHORDS = CHORDS;

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 14
      INT16 - QSTRTC=152
      INT8  - 1
      BBOOL - .TRUE.
      INT8  - ORDER
      BBOOL - .FALSE.
```

```
INT8   - DIMEN+1
BBOOL  - (.NOT.OPNCLS)
BBOOL  - (.NOT.NONPER)
BBOOL  - .FALSE.
INT32  - NVERT

  REPEAT NVERT TIMES
INT16  - 34
INT16  - QCRVEC=296
PSREAL- V (1,1)
PSREAL- V (2,1)
PSREAL- V (3,1)
PSREAL- V (4,1)

(OPTIONAL)
  REPEAT NKNOTS TIMES
INT16  - 10
INT16  - QKNOT=295
PSREAL- KNOTS (I)

INT16  - 14
INT16  - QENDCV=153
INT32  - CHORDS
PSREAL- 0
```

REMOVE HANDLE;

```
INT16  - 2
INT16  - QREMOB=119
INT16  - SIZE+8
INT16  - QNAME=45
INT16  - SIZE
INT16  - 1
ID     - HANDLE, SIZE
INT16  - 0
```

REMOVE FOLLOWER OF HANDLE;

```
INT16  - 2
INT16  - QUNFOL=117
INT16  - SIZE+8
INT16  - QNAME=45
INT16  - SIZE
INT16  - 1
ID     - HANDLE, SIZE
INT16  - 0
```

```
REMOVE HANDLE1 FROM HANDLE2;

        INT16 - 2
        INT16 - QSETRM=124
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE1, SIZE
        INT16 - 0
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE2, SIZE
        INT16 - 0


REMOVE PREFIX OF HANDLE;

        INT16 - 2
        INT16 - QUNPFX=116
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0


HANDLE := ROTATE IN X ANGLE (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 10
        INT16 - QROTX=74
        PSREAL- ANGLE
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
HANDLE := ROTATE IN Y ANGLE (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 10
      INT16 - QROTY=75
      PSREAL- ANGLE
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0

HANDLE := ROTATE IN Z ANGLE (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 10
      INT16 - QROTZ=76
      PSREAL- ANGLE
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0

HANDLE := RATIONAL POLYNOMIAL
        ORDER = ORDER
        (DIMENSION IMPLIED IN SYNTAX)
        COEFFICIENTS = X(I),   Y(I),   Z(I),   W(I)
                       X(I-1), Y(I-1), Z(I-1), W(I-1)
                          :       :       :       :
                       X(0),   Y(0),   Z(0),   W(0)
        CHORDS = CHORDS;

      INT16 - SIZE+8
      INT16 - QLABEL=44
```

```
INT16 - SIZE
INT16 - 1
ID    - HANDLE, SIZE
INT16 - 0
INT16 - 14
INT16 - QSTRTC=152
INT8  - 2
BBOOL - .TRUE.
INT8  - ORDER
BBOOL - .FALSE.
INT8  - DIMEN+1
BBOOL - (.TRUE.)
BBOOL - (.TRUE.)
BBOOL - .FALSE.
INT32 - ORDER+1

  REPEAT ORDER+1 TIMES
INT16 - 34
INT16 - QCRVEC=296
PSREAL- V (1,1)
PSREAL- V (2,1)
PSREAL- V (3,1)
PSREAL- V (4,1)

INT16 - 14
INT16 - QENDCV=153
INT32 - CHORDS
PSREAL- 0
```

RESERVE_WORKING_STORAGE Bytes;

```
INT16 - 6
INT16 - QRSVST=314
INT32 - BYTES
```

HANDLE := SCALE BY X,Y,Z (APPLIED TO APPLY);

```
INT16 - SIZE+8
INT16 - QLABEL=44
INT16 - SIZE
INT16 - 1
ID    - HANDLE, SIZE
INT16 - 0
INT16 - 26
INT16 - QSCALE=164
PSREAL- X(1)
PSREAL- Y(2)
```

```
      PSREAL- Z(3)
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0


HANDLE := SET CONDITIONAL_BIT BITNUM ONOFF
        (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 6
      INT16 - QSETBT=89 OR QCLRBT=90
      INT32 - BITNUM
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0


HANDLE := SET CHARACTERS SCREEN_ORIENTED/FIXED
        (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 6
      INT16 - QCHARP=253
      INT32 - 1
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0
```

```
HANDLE := SET CHARACTERS SCREEN_ORIENTED
        (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 6
      INT16 - QCHARP=253
      INT32 - 0
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0

HANDLE := SET CHARACTERS WORLD_ORIENTED
        (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 6
      INT16 - QCHARP=253
      INT32 - -1
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0

SETUP CNESS TRUE/FALSE <INP>HANDLE;

      INT16 - SIZE + 12
      INT16 - QCNESS=330
      INT16 - INP
      INT16 - 0 OR 1
      INT16 - 0
      INT16 - SIZE
      ID    - HANDLE, SIZE
      INT16 - 0
```

```
HANDLE := SET COLOR HUE,SAT (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 18
      INT16 - Q2COLR=167
      PSREAL- HUE
      PSREAL- SAT
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0


HANDLE := SET CONTRAST TO CONTRAST
        (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 10
      INT16 - QCONTR=232
      PSREAL- CONTRA
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0

HANDLE := SECTIONING_PLANE (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 2
```

```
        INT16 - QSECPL=315
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0


HANDLE := SET DISPLAYS ALL ONOFF (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 4
        INT16 - QSCOPS=93
        BOOL  - ONOFF
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0


HANDLE := SET DEPTH_CLIPPING ONOFF (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 4
        INT16 - QDCLIP=95
        BOOL  - ONOFF
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
HANDLE := SET DISPLAY N ONOFF (APPLIED TO APPLY);

       INT16 - SIZE+8
       INT16 - QLABEL=44
       INT16 - SIZE
       INT16 - 1
       ID    - HANDLE, SIZE
       INT16 - 0
       INT16 - 6
       INT16 - QSTDSP=235
       INT32 - N
       INT16 - 2
       INT16 - QDSCON=233 OR QDSCOF=234
       INT16 - SIZE+8
       INT16 - QNAME=45
       INT16 - SIZE
       INT16 - 1
       ID    - APPLY, SIZE
       INT16 - 0


HANDLE := SET INTENSITY ONOFF IMIN:IMAX
          (APPLIED TO APPLY);

       INT16 - SIZE+8
       INT16 - QLABEL=44
       INT16 - SIZE
       INT16 - 1
       ID    - HANDLE, SIZE
       INT16 - 0
       INT16 - 20
       INT16 - QSTINT=301
       BOOL  - ONOFF
       PSREAL- IMIN
       PSREAL- IMAX
       INT16 - SIZE+8
       INT16 - QNAME=45
       INT16 - SIZE
       INT16 - 1
       ID    - APPLY, SIZE
       INT16 - 0


HANDLE := SET LINE_TEXTURE PATTRN <AROUND> (APPLIED TO APPLY);

       INT16 - SIZE+8
       INT16 - QLABEL=44
       INT16 - SIZE
```

```
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QTXTUR=344 OR QCTXTR=345
        INT32 - PATTRN
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
HANDLE := SET LEVEL_OF_DETAIL TO LEVEL
        (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QLEVEL=88
        INT32 - LEVEL
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
HANDLE := SET PICKING IDENTIFIER = PICKID
 (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 2
        INT16 - QPCKNM=110
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
```

```
      INT16 - 1
      ID    - PICKID, SIZE
      INT16 - 0
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0


HANDLE := SET PICKING LOCATION = XCENTR, YCENTR
                                 XSIZE,  YSIZE
  (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 50
      INT16 - QPCKBX=194
      PSREAL- XCENTR
      PSREAL- YCENTR
      INT32 - 0
      INT32 - 0
      INT32 - 0
      INT32 - 0
      PSREAL- XSIZE
      PSREAL- YSIZE
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0


HANDLE := SET PICKING ONOFF (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 8
```

```
        INT16 - QPCKNG=91
        BOOL  - ONOFF
        INT32 - 0
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0


HANDLE := SET RATE PHASEON PHASEOFF INITIAL STATE DELAY
        (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 34
        INT16 - QBLDEF=205
        PSREAL- PHASEON
        PSREAL- PHASEOFF
        PSREAL- INITIAL STATE (1-ON OR 0-OFF)
        PSREAL- DELAY
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0


HANDLE := SET RATE EXTERNAL (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QSETBI=89
        INT32 - 15
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
```

```
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0


SEND TRUE/FALSE  TO <INP> DEST;

      INT16 - 4
      INT16 - QBOOL=2
      BOOL  - B

      INT16 - SIZE+8
      INT16 - QALOOK=100
      INT16 - SIZE
      INT16 - 1
      ID    - DEST, SIZE
      INT16 - 0
      INT16 - 6
      INT16 - QINPIN=145
      INT32 - INP
      INT16 - 2
      INT16 - QSTORE=137


SEND FIX (I) TO <INP> DEST;

      INT16 - 6
      INT16 - QINTGR=3
      INT32 - I
      INT16 - SIZE+8
      INT16 - QALOOK=100
      INT16 - SIZE
      INT16 - 1
      ID    - DEST, SIZE
      INT16 - 0
      INT16 - 6
      INT16 - QINPIN=145
      INT32 - INP
      INT16 - 2
      INT16 - QSTORE=137


SEND M2D (MAT) TO <INP> DEST;

      INT16 - 50
      INT16 - QM2BLD=254
      PSREAL- MATRIX (1,1)
      PSREAL- MATRIX (1,2)
```

```
        PSREAL- 0
        PSREAL- 0
        PSREAL- MATRIX (2,1)
        PSREAL- MATRIX (2,2)
        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - DEST, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QINPIN=145
        INT32 - INP
        INT16 - 2
        INT16 - QSTORE=137


SEND M3D (MAT) TO <INP> DEST;

        INT16 - 90
        INT16 - QM3BLD=255
        PSREAL- MATRIX (1,1)
        PSREAL- MATRIX (1,2)
        PSREAL- MATRIX (1,3)
        PSREAL- 0
        PSREAL- MATRIX (2,1)
        PSREAL- MATRIX (2,2)
        PSREAL- MATRIX (2,3)
        PSREAL- 0
        PSREAL- MATRIX (3,1)
        PSREAL- MATRIX (3,2)
        PSREAL- MATRIX (3,3)
        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - DEST, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QINPIN=145
        INT32 - INP
        INT16 - 2
        INT16 - QSTORE=137
```

```
SEND M4D (MAT) TO <INP> DEST;

        INT16 - 130
        INT16 - QM4BLD=256
        PSREAL- MATRIX (1,1)
        PSREAL- MATRIX (1,2)
        PSREAL- MATRIX (1,3)
        PSREAL- MATRIX (1,4)
        PSREAL- MATRIX (2,1)
        PSREAL- MATRIX (2,2)
        PSREAL- MATRIX (2,3)
        PSREAL- MATRIX (2,4)
        PSREAL- MATRIX (3,1)
        PSREAL- MATRIX (3,2)
        PSREAL- MATRIX (3,3)
        PSREAL- MATRIX (3,4)
        PSREAL- MATRIX (4,1)
        PSREAL- MATRIX (4,2)
        PSREAL- MATRIX (4,3)
        PSREAL- MATRIX (4,4)

        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - DEST, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QINPIN=145
        INT32 - INP
        INT16 - 2
        INT16 - QSTORE=137


SEND COUNT*DRAWMV TO <INP> DEST;

        INT16 - 6
        INT16 - QNBOOL=243
        BOOL  - DRAWMV
        INT16 - COUNT
        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - DEST, SIZE
        INT16 - 0
        INT16 - 6
```

```
      INT16 - QINPIN=145
      INT32 - INP
      INT16 - 2
      INT16 - QSTORE=137


SEND REAL-NUMBER TO <INP> DEST;

      INT16 - 10
      INT16 - QREAL=4
      PSREAL- R
      INT16 - SIZE+8
      INT16 - QALOOK=100
      INT16 - SIZE
      INT16 - 1
      ID    - DEST, SIZE
      INT16 - 0
      INT16 - 6
      INT16 - QINPIN=145
      INT32 - INP
      INT16 - 2
      INT16 - QSTORE=137


SEND 'STR' TO <INP> DEST;

      INT16 - SIZE + 6
      INT16 - QSTR=5
      INT16 - SIZE
      INT16 - 1
      STRING- STR, SIZE
      INT16 - SIZE+8
      INT16 - QALOOK=100
      INT16 - SIZE
      INT16 - 1
      ID    - DEST, SIZE
      INT16 - 0
      INT16 - 6
      INT16 - QINPIN=145
      INT32 - INP
      INT16 - 2
      INT16 - QSTORE=137

SEND V2D (V) TO <INP> DEST;

      INT16 - 34
      INT16 - QVEC2=10
      PSREAL- V (1)
```

```
        PSREAL- V (2)
        PSREAL- 0
        PSREAL- 0
        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - DEST, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QINPIN=145
        INT32 - INP
        INT16 - 2
        INT16 - QSTORE=137


SEND V3D (V) TO <INP> DEST;

        INT16 - 34
        INT16 - QVEC3=13
        PSREAL- V (1)
        PSREAL- V (2)
        PSREAL- V (3)
        PSREAL- 0
        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - DEST, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QINPIN=145
        INT32 - INP
        INT16 - 2
        INT16 - QSTORE=137


SEND V4D (V) TO <INP> DEST;

        INT16 - 34
        INT16 - QVEC4=16
        PSREAL- V (1)
        PSREAL- V (2)
        PSREAL- V (3)
        PSREAL- V (4)
        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - DEST, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QINPIN=145
        INT32 - INP
        INT16 - 2
        INT16 - QSTORE=137
```

```
SEND VALUE (VARNAM) TO <INP> DEST;

        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - VARNAM, SIZE
        INT16 - 0
        INT16 - 2
        INT16 - QFETCH=186
        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - DEST, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QINPIN=145
        INT32 - INP
        INT16 - 2
        INT16 - QSTORE=137
```

```
SEND VL (HANDLE1) TO <INP> HANDLE2;

        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE1, SIZE
        INT16 - 0
        INT16 - SIZE+8
        INT16 - QALOOK=100
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE2, SIZE
        INT16 - 0
        INT16 - 6
        INT16 - QINPIN=145
        INT32 - INP
        INT16 - 2
        INT16 - QSTORE=137


HANDLE := SOLID_RENDERING (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 2
        INT16 - QSOLRE=343
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0

HANDLE := STANDARD FONT (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 2
        INT16 - QSTDFO=132
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0

HANDLE := SURFACE_RENDERING (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 2
        INT16 - QSURRE=342
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
HANDLE := TEXT SIZE SIZEX SIZEY (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 18
        INT16 - QTEXTS=339
        PSREAL-  SIZEX
        PSREAL-  SIZEY
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
HANDLE := TRANSLATE BY V (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 26
        INT16 - QTRANS=73
        PSREAL-  V(1)
        PSREAL-  V(2)
        PSREAL-  V(3)
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
VARIABLE HANDLE;

      INT16 - SIZE+8
      INT16 - QVARNM=204
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0


HANDLE := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE) N=N
         <VECTORS>;

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 45
      INT16 - Q2DVHD=147 OR Q3DVHD=148
      INT8  - 1 (DOTS)
   OR
      INT8  - 3 (CONNECTED, ITEMIZED)
   OR
      INT8  - 4 (SEPARATE)
      BBOOL - BNORM
      INT32 - 0
      PSREAL- 0
      PSREAL- 0
      PSREAL- 0
      PSREAL- 0
      INT32 - VECCOU
      BBOOL - CBLEND
   BLOCK NORMALIZED
      INT16 - 4+COUNT*2*DIMEN+2
      INT16 - QBNDAT=266
      INT16 - COUNT*2*DIMEN+2
      VBLNO - VECS, POSLIN, DIMEN, COUNT
   VECTOR NORMALIZED
      INT16 - 4+COUNT*(DIMEN+1)*2
      INT16 - QBNDAT=266
      INT16 - COUNT*(DIMEN+1)*2
      VECNO - VECS, POSLIN, DIMEN, COUNT
      INT16 - 2
      INT16 - QENDLS=107
```

```
HANDLE := VIEWPORT HORIZONTAL = XMIN:XMAX
          VERTICAL   = YMIN:YMAX
          INTENSITY  = IMIN:IMAX
          (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 54
      INT16 - QVIEW=160
      PSREAL- XMIN
      PSREAL- XMAX
      PSREAL- YMIN
      PSREAL- YMAX
      PSREAL- IMIN
      PSREAL- IMAX
      INT32 - 0
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0


HANDLE := WINDOW X = XMIN:XMAX
          Y = YMIN:YMAX
          FRONT BOUNDARY = FRONT
          BACK  BOUNDARY = BACK
          (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 50
      INT16 - QWINDO=157
      PSREAL- XMIN
      PSREAL- XMAX
      PSREAL- YMIN
      PSREAL- YMAX
      PSREAL- FRONT
```

```
        PSREAL- BACK
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0


HANDLE := WRITEBACK (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 2
        INT16 - QWBACK=277
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0


HANDLE := CANCEL XFORM (APPLIED TO APPLY);

        INT16 - SIZE+8
        INT16 - QLABEL=44
        INT16 - SIZE
        INT16 - 1
        ID    - HANDLE, SIZE
        INT16 - 0
        INT16 - 2
        INT16 - QXFCAN=273
        INT16 - SIZE+8
        INT16 - QNAME=45
        INT16 - SIZE
        INT16 - 1
        ID    - APPLY, SIZE
        INT16 - 0
```

```
HANDLE := XFORM MATRIX (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 2
      INT16 - QXFMAT=270
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0


HANDLE := XFORM VECTOR_LIST (APPLIED TO APPLY);

      INT16 - SIZE+8
      INT16 - QLABEL=44
      INT16 - SIZE
      INT16 - 1
      ID    - HANDLE, SIZE
      INT16 - 0
      INT16 - 2
      INT16 - QXFVEC=271
      INT16 - SIZE+8
      INT16 - QNAME=45
      INT16 - SIZE
      INT16 - 1
      ID    - APPLY, SIZE
      INT16 - 0
```

3. Description of Six-Bit Binary Data Protocol in the PS 390

The following sections describe how the PS 390 binary data can be encoded into bytes with six bits of binary data per byte. This method should be used when you need to transmit printable ASCII characters to the PS 390.

3.1. Data Storage

The PS 390 stores its data as follows:

MSB	LSB

where the MSB starts at the low address and the LSB starts at the high address.

The host must send the MSB first, followed by the LSB. Some hosts store their data in an address order that reverses this sequence. If this is the case, the MSB and LSB must be reversed in the host before being sent to the PS 390.

3.2. Six-Bit Binary Data Encoding Method

Binary data must be encoded in the following manner. This encoding process occurs prior to sending the byte count of binary vector data.

NOTE

The byte count of binary data must not include the count of bytes that result when the data is passed through the encoding scheme.

1. The encoding process collects 16-bit words until it has two sets.

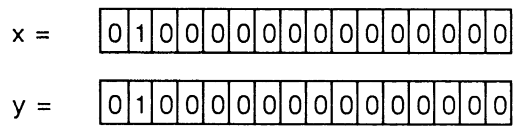
CAUTION

The carriage control characters must be suppressed when transmitting binary data, or the carriage control characters will be interpreted as binary data.

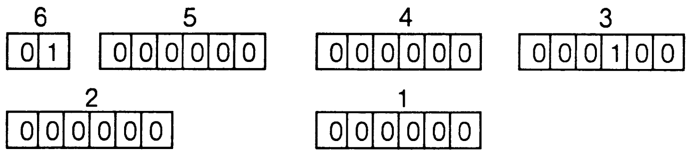
2. A two-word set is broken up into five bytes with six significant bits, and one byte with two significant bits. These bits are extracted from the least significant end to the most significant end of the two-word set.

3. The order that the bytes are sent to the PS 390 reverses the order in which they were extracted. The byte with two significant bits is sent first, followed by the the last 6-significant-bit byte, and so on.

4. To make the bytes printable ASCII characters, a zero '0' character (hex 30 or decimal 48) is added to each byte prior to sending them. This encoding process is illustrated in the example that follows. The two-word set of 16-bits are held internally in the host in the following bit sequence. The first two word set is:



The two 16-bit words are broken up into five bytes with six bits, and one byte with two bits in the following order:



The zero "0" character is now added to each six-bit byte:

```
byte1       000000                byte4       000000
          + 110000 (zero)                   + 110000 (zero)
            110000                            110000


byte2       000000                byte5       000000
          + 110000 (zero)                   + 110000 (zero)
            110000                            110000


byte3       000100                byte6       ----01
          + 110000 (zero)                   + 110000 (zero)
            110100                            110001
```

After the encoding procedure, the bits will be sent to the PS 390 in the following sequence of bytes. Note that the sequence order of the bytes has been reversed.

byte6	Two significant bits	0 0 1 1 0 0 0 1
byte5	Six significant bits	0 0 1 1 0 0 0 0
byte4	Six significant bits	0 0 1 1 0 0 0 0
byte3	Six significant bits	0 0 1 1 0 1 0 0
byte2	Six significant bits	0 0 1 1 0 0 0 0
byte1	Six significant bits	0 0 1 1 0 0 0 0

3.3. Example of Encoding Binary Data

An example of encoding binary vector data is given in the following section. Please note that the example assumes escape mode. Refer to Section *RM5, Host Communications*, for a complete description of escape mode.

The vector list to be encoded is:

```
AA:=  vec itemized n=4
      P 1,1,0 I=1.0
      L -.25, .75, .5 I= .75
      P 10,5,.001  I=.5
      L -.001, -.002, .003 I= .1
      ;
```

The data in PS 390 Eight-bit binary format is as follows:

NOTE

The X,Y,Z mantissa's and the Vector exponent are two's complement numbers.

```
0100000000000000  = 1/2 (x mantissa)
0100000000000000  = 1/2 (y mantissa)
0000000000000000  = 0   (z mantissa)
00000001 1111111 0 exponent =1 intensity= 7F(hex) p/l=p

1110000000000000  = -1x2**-2 (x mantissa) NOTE 2's complement
0110000000000000  = 3x2**-2  (y mantissa)
0100000000000000  = 1x2**-1  (z mantissa)
00000000 1100000 1 exponent=0 inten=60(hex) p/l=l

0101000000000000  = 5x2**-3
0010100000000000  = 5x2**-4
0000000000000010  = 1x2**-14
00000100 1000000 0 exp=4 int=40(hex) p/l=p

1101111100111100  = -2097x2**-13
1011111001110111  = -16777x2**-15
0110001001001101  =  25145x2**-15
11111000 0001100 1 exp= -8 int=0C(hex) p/l=l

0000000000000000    padding
```

For exercise, check the last vectors in decimal.

```
x= -2097x2**-13x2**-8 = -2097x2**-21 = -9.99928E -4 -> -.001

y= -1677x2**-15x2**-8 = -16777x2**-23 = 1.99997E -3 -> -.002

z= 25145x2**-15x2**-8 = 25145x2**-23  = 2.99752E -3 ->  .003
```

The left column is the binary data re-encoded into the six-bit format. The right column is the eight-bit data.

```
002P0\    0000000000001010         = 10 (size of label+8)
          0000000000101100         = 44 (Qlabel)

000P01    0000000000000010         = 2 (size)
          0000000000000001         = 1

11@@00    0100000101000001         = AA (name)
          0000000000000000         = 0

00;@2D    0000000000101101         = 45 (data byte count)
          0000000010010100         = 148 (q3dvhd)

030000    0000001100000000         item= 3/bnorm= .false.
          0000000000000000         = 0 (4 byte integer)

000000    0000000000000000
          0000000000000000         = 0 (8 byte real)

000000    0000000000000000
          0000000000000000

000000    0000000000000000
          0000000000000000         = 0 (8 byte real)

000000    0000000000000000
          0000000000000000

000000    0000000000000000
          0000000000000000         = 0 (8 byte real)

000000    0000000000000000
          0000000000000000

000000    0000000000000000
          0000000000000000         = 0 (8 byte real)

000000    0000000000000000
          0000000000000000

000000    0000000000000000
          0000000000000000         = 4 (veccount 4 byte integer)
```

```
001000    0000000000000100
          00000000                      = .false. (cblend)
          00000000

0T0@X0    00100100                      = 36 (total byte count)
          0000000100001010              = 266 (qbndat)
          00000000

0P@010    00100000                      = 32 (count*(dimen+1)*2)
          0100000000000000              = 1/2(x mantissa)
          01000000

000001    00000000                      = 1/2(y mantissa)
          0000000000000000              = 0 (z mantissa)
          00000001                        exponent=1

3nh01P    1111111 0                       intensity=7F(hex) p/l=p
          1110000000000000              = -1*x**-2 (x mantissa)
          01100000

00@000    00000000                      = 3*2**-1 (y mantissa)
          0100000000000000              = 1*2**-1 (z mantissa)
          00000000                        exponent=0

31D00X    1100000 1                       intensity=60(hex) p/l=l
          0101000000000000              = 5*2**-3 (x mantissa)
          00101000

000084    00000000                      = 5*2**-4 (y mantissa)
          0000000000000010              = 1*2**-14(z mantissa)
          00000100                        exponent=4

20gcbn    1000000 0                       intensity=40(hex) p/l=p
          1101111100111100              = -2097*2**-13 (x mantissa)
          10111110

1gHTgh    01110111                      = -16777*2**-15 (y mantissa)
          0110001001001101              = 25145*2**-15(z mantissa)
          11111000                        exponent=-8

0I00D0    0001100 1                       intensity=0C(hex) p/l=l
          0000000000000101              = 5 (data byte count - including
                                             padding)
          00000000

1[0000    01101011                      = 107 (qendls)
          00000000                        padding
          0000000000000000

002P0\000P0111@@0000;@2D03000000000000000000000000000000000000000000000
000000000
00000010000T0@X00P@0100000013nh01P00@00031D00X00008420gcbn1gHTgh0I00D01
[0000
```

The vector list is transmitted to the PS 390 using escape mode as follows:

```
<SOP>2 {route to six bit binary route}
{and send the encoded data}
```

If the vectors are vector normalized, an arbitrary sized vector list may be transmitted by sending multiple packets of data as above, without the terminating semicolon. When all of the vectors have been transmitted, they must be terminated with a semicolon. However the packets must contain complete vectors.

The vectors will be processed faster if the vector estimate (N=vector_estimate) is equal to or greater than the actual number of vectors transmitted. This is because the PS 390 allocates memory for the vectors based on this estimate. If the estimate is low, the PS 390 must find a new block of continuous memory large enough for the total, copy the vectors in the original block into the new block, and write the new vectors into the block.

PS 390 RELEASE NOTES

EVANS & SUTHERLAND

April 1987
E&S #904015-605
Rev B

CONTENTS

FIGURES AND TABLES

PS 390 GRAPHICS FIRMWARE RELEASE NOTES
Version A2.V02

1. GENERAL INTRODUCTION

These release notes document functionality of the PS 390 and are intended as a supplement to the *PS 300 Document Set* which describes the operation and programming of the PS 300 line of computer graphics systems. These notes can be placed in the *Document Set* behind the Release Notes tab in Volume 3A.

The PS 390 has a new hardware configuration with a reduced-size cabinet and a Joint Control Processor (JCP) card, which are explained in Section 2.1 of these Notes. The hardware configuration and calligraphic display documented in the *Document Set* are not applicable to the PS 390. New users should note that a different set of peripherals is available with the PS 390 although the peripherals documented in the *Document Set* are also supported under PS 390. Procedures for using the peripherals with the multiplexing box are explained in Section 2.3.

You should assume that all programming information in the *PS 300 Document Set* and accompanying installation manuals (dependent on your particular configuration) is applicable to the PS 390 unless specifically noted in these Release Notes.

Changes and additions to the PS 300 Graphics runtime firmware and host software released since the publication of the *Document Set* are contained in Part II of these Notes. This section consolidates the information from previous release notes that applies to the PS 390, and includes formal change pages for the Command and Function Summaries and for the Graphics Support Routines (GSRs) in the *PS 300 Document Set*. Please discard the old pages in your set and replace with these new pages. Some new pages documenting specific PS 390 functionality and other pages with new information are included and should be inserted in the appropriate place in your *Document Set*.

1.1 Notes to New Users

New users should familiarize themselves with the information in these notes and in the *PS 300 Document Set* and note where PS 390 information contained in this package differs from information in the *PS 300 Document Set*.

1.2 Notes to Current Users

One of the primary concerns in developing the PS 390 runtime firmware was maintaining compatibility with previous systems so that existing PS 300 programs would run on the PS 390 without modification. This was almost completely achieved. However, some incompatibilities exist because Port 2 is no longer used and there is no support for DMR-11 interface or multi-user systems. Also, scope 0 is the only scope enabled; therefore, Set Scope commands (both ASCII and GSR) should not be used.

The PS 390 new hardware configuration with a reduced-size cabinet and a Joint Control Processor (JCP) card is explained in Section 2.1 of these Notes. If you have already upgraded to a reduced-size cabinet, you should have already received this information.

You should also have Release Notes for Version A1.V02 and Version A2.V01 of the graphics runtime firmware. Note that the information contained in those release notes which reflects current functionality has been consolidated and included in Part II of these notes.

Current PS 340 users should note that rendering capability on the PS 390 is available only by ordering the rendering option. Without this option, you can display objects defined as polygons on the PS 390, but you cannot perform any rendering operations. For users with the rendering option, the PS 390 Rendering Option Release Notes are included with the PS 390 Release Notes package.

Current PS 350 users should note that the PS 390 is plug compatible with PS 350 applications with the exceptions given in Section 4 of these Notes. Please note that the light pen is not supported with this release. The nodes created by applications using the Lightpen command (both ASCII and GSR) will be treated as no-operation nodes.

Some PS 390 information contained in this release package is identical to information and change pages contained in the *PS 350 User's Manual.* Please note that information in these release notes supersedes any other documentation explaining similar or identical capabilities of the PS 390.

1.3 Notes to All Users

As previously mentioned, a different set of peripherals is offered with the PS 390 although existing peripherals are still supported. Users of existing peripherals and users with new peripherals must both use a multiplexing box as there is no data concentrator on the display. Procedures for using the new mux boxes are given in Section 2.3 of these Notes. Documentation for the new set of peripherals is supplied as a separate document included with this release package.

Data formats provided for those users with the Parallel Interface or for those who access internal data are provided in the *PS 300 Advanced Programming* guide included with this release package. (PS 350 users please note that PS 390 data formats and PS 350 data formats are identical). No new GSRs routines are provided with this release. GSR routines supporting new PS 390 capability are planned for future releases.

Please take special note of Section 4 of these Release Notes, which documents PS 390 exceptions to existing PS 300 documentation.

Direct your questions and comments to the Evans & Sutherland Customer Engineering Hotline 1-800-582-4375 (except Utah). Within Utah, customers should call 582-9412.

1.4 Release Package Contents

This PS 390 Release Package contains the following items.

- One copy of the Graphics runtime firmware Version A2.V02

  For users with the rendering option, this is on the Visualization diskette. Instructions for loading the firmware are contained in Volume 5 of the *Document Set*. Instructions for configuring your firmware diskette according to which options you have at your installation are contained in section 3 of these Notes.

- PS 390 host software distributed on magnetic tape including (but not limited to) the following:

  - PS 300 Graphics Support Routines (GSRs). The files READFOR.GSR and READPAS.GSR contain descriptions of the FORTRAN and Pascal GSR software.

  - The PS 300 Host-Resident I/O Subroutines

  - Three programming utilities: NETEDIT, NETPROBE, and MAKEFONT (For VAX/VMS users only).

  - Writeback Feature

  Documentation for the Writeback feature is included in this release package. More detail on the GSRs, I/O Subroutines, and programming utilities can be found in Volumes 3 and 4 of the *Document Set*.

- One copy of the Diagnostic Utility Diskette

  This diskette provides all the utility programs described in Volume 5, Section 10 of the *Document Set*. Please refer to that section for instructions on using the utility programs for backup and file management and make note that the new Diagnostic Utility Diskette is the only diskette that should be used to load these programs.

- *PS 390 Raster Programming* guide

  This manual documents how to send run-length encoded pixel data to the PS 390.

- *PS 300 Advanced Programming* guide

  This manual is intended for use by experienced programmers as a guide to writing functions and as a reference for doing direct Physical I/O with the Parallel Interface.

- *PS 390 Peripherals Reference Manual*

1.5 Distribution Tape Format and Installation Procedure

All PS 390 VAX/VMS sites will receive the distribution tape (PS 390 host software) in VMS Backup format. To install the VAX PS 390 host software, first create a subdirectory for the PS 390 software and set your default to that directory. Using the VMS Backup Utility, enter the following commands:

```
$  Allocate MTNN:
$  Mount/Foreign MTNN:
$  Backup MTNN:PSDIST.BCK [...]*.*
$  Dismount MTNN:
$  Deallocate MTNN:
```

where MTNN: is the physical device name of the tape drive being used.

This will create the subdirectory A2V01.DIR which is the parent directory of the PS 390 host software.

UNIX sites will receive a 1600-bpi distribution tape in tar format. IBM sites will receive a 1600-bpi distribution tape with a block size of 6400 and a logical record length of 80.

All PS 390 sites that are not DEC VAX/VMS, UNIX, or IBM, will receive a variable length ANSI format distribution tape containing the PS 390 host software. Consult your system operation manual for instructions on reading ANSI-formatted tapes.

2. INTRODUCTION TO PS 390

The PS 390 provides the real–time interaction capability and line quality of a calligraphic system with the flicker–free images of a raster system, combining the desirable features of both technologies while eliminating the disadvantages of each.

These capabilities were accomplished by the development of several VLSI chips and one custom gate array designed for the real–time manipulation of anti–aliased raster lines matching or exceeding the quality of calligraphic lines.

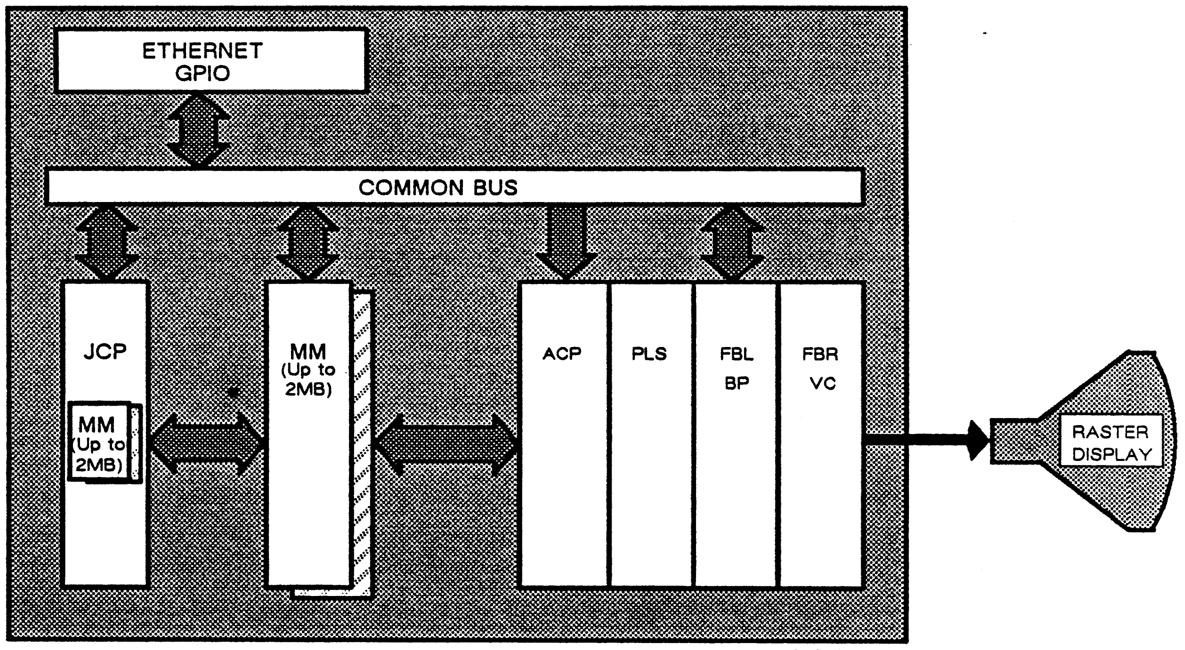
The graphics pipeline of the PS 390 is 32 bits which provides high–precision processing required for large and complex models. The frame buffer is a 48–bit frame buffer, double buffered.

With this version of the firmware, you can use the PS 390 monitor to display host–generated pixel images. The PS 390 accepts raster data in run–length encoded format. A discussion of how to accomplish this is contained in the *PS 390 Raster Programming* manual included with this release package. Existing PS 340 applications using run–length encoding to display host–generated images will run unchanged on the PS 390.

The local capability to create, render, and shade polygonal models on the PS 390 is available with the purchase of the rendering option. With this option, you can display and manipulate a wireframe model in one viewport of the screen and display the same model as a shaded image in another viewport on the screen. Capability now supported on the PS 340 graphics system is supported on the PS 390 with the rendering option. This includes the ability to apply sectioning, back–face removal, and hidden–line rendering operations to wireframe models and to display static images with a wash, flat, Phong, or Gouraud shading style.

2.1 System Hardware Overview

The PS 390 is housed in a new reduced size cabinet and contains a new Joint Control Processor (JCP) card. The JCP replaces the Graphics Control Processor, up to two mass memory cards, and (optionally) the PS 300 IBM 3278 GPIO card. The description of PS 300 Control Unit in the *Document Set* is for systems with a larger cabinet and a GCP. The PS 390 has six basic circuit cards: JCP, Mass Memory (MM), Arithmetic Control Processor (ACP), Pipeline Subsystem (PLS), Frame Buffer and Bit–Slice Processor (FBL/BP), and Frame buffer and Video Controller (FBR/VC). The architecture for the PS 390 is shown in Figure 1.



IAS390002P3

JCP	-	JOINT (GRAPHICS) CONTROL PROCESSOR	FBL	-	FRAME BUFFER LEFT
MM	-	MASS MEMORY (1- to 4-MBYTES)	BP	-	BITSLICE PROCESSOR
ACP	-	ARITHMETIC CONTROL PROCESSOR	FBR	-	FRAME BUFFER RIGHT
PLS	-	PIPELINE SUBSYSTEM	VC	-	VIDEO CONTROLLER
			GPIO	-	GENERAL PURPOSE INTERFACE OPTION

Figure 1. PS 390 Architectural Overview

The free-standing control unit of the PS 390 requires no clearance for operation, provided that site-specific heat dissipation requirements are met. It is mounted on casters for easy portability and to provide return air to the unit fan.

The control unit is approximately 53 cm (21 inches) wide, 71 cm (28 inches) deep, 67 (26.5 inches) high, and weighs 55 kg (120 pounds). The top holds over 250 pounds static weight; 180 pounds rolling load. (See Figure 2.)

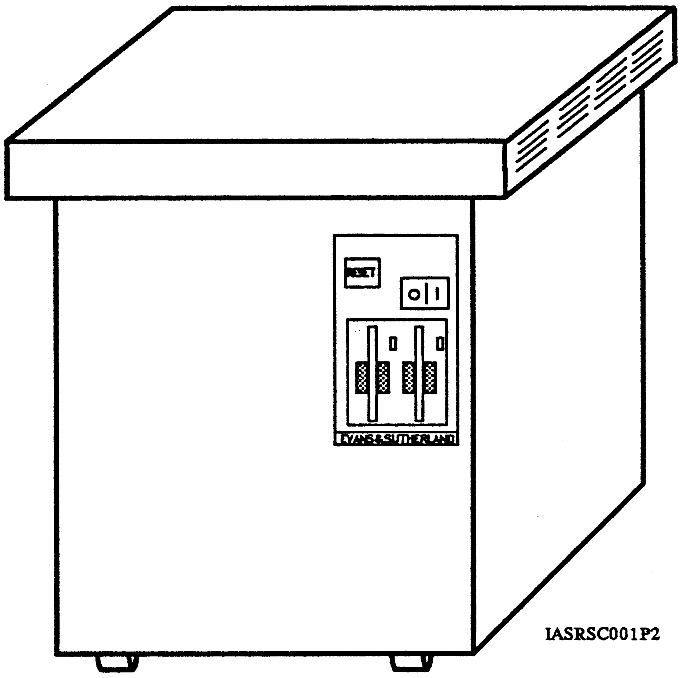


Figure 2. PS 390 Control Unit

There are two external controls on the PS 390 control unit. One is the ON/OFF circuit breaker switch, located at the top right of the front panel. This switch is recessed and surrounded by a protective frame. A RESET switch is located just left of the circuit breaker. The RESET switch allows the system to be reset instead of powered off during a system lock or reboot.

The PS 300 floppy disk drives are located at the front of the unit near the upper, right corner.

The PS 390 uses a double-sided, quad-density, 5-1/4 inch minifloppy diskette capable of storing 737,280 formatted data bytes on 160 tracks.

At the back of the control unit, above the power distribution panel, is the communications connector panel. See Figure 3. The panel is vertically aligned, with ports 0-5 from the top down. Connectors are externally accessible on the back of the control unit.

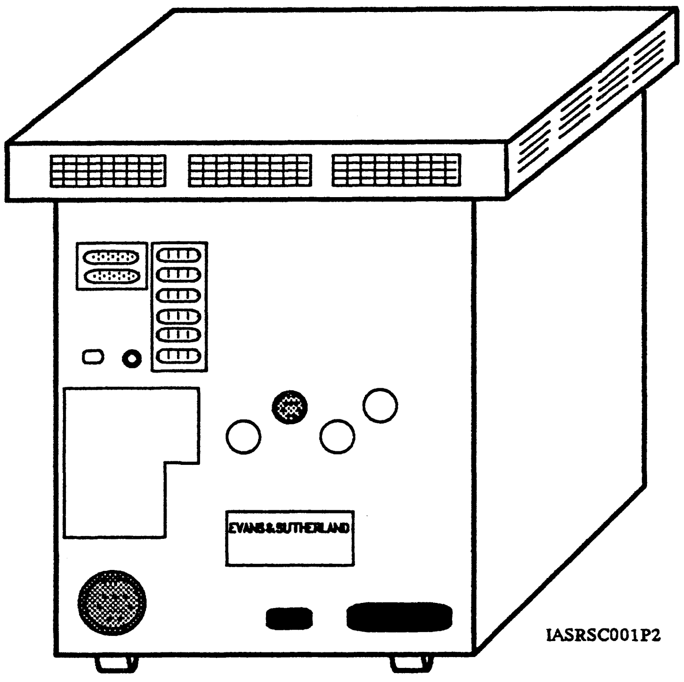


Figure 3. Back of Control Unit

The standard PS 390 control unit comes with the cards in place. The metal casing on the inside of the unit replaces the Faraday cage installed in some older cabinets.

The PS 390 is FCC Class A certified for emissions and will meet UL 478 and CSA 22.2 #154 safety standards.

The new Joint Control Processor (JCP) card consists of two (optionally three) sections: Control Processor, Mass Memory, and Interface section.

The control processor (CP) section is functionally similar to the old graphics control processor (GCP) and is based on a 68000 10 MHz microprocessor.

This differs from the GCP card documented in the Document Set in that:

- Local memory is increased from 256K to 512K.
- There is a local path to the JCP resident mass memory that is used instead of the Common bus path (GCP systems) thus providing faster access to mass memory from the 68000.
- Four usable asynchronous RS-232 ports are supported (compared to five on the GCP) which reside on the Communications Connector Panel. See Figure 4. Port 0 and Port 2 are physically present but not usable. This means that the DMR-11 interface is not available with reduced size cabinet systems nor is multi-user functionality.

The new port configuration for the PS 390 is as follows:

Port 1 is the host port.

Port 3 is the debug port, for diagnostic purposes.

Port 4 may be used for special interface applications, including an alternate diagnostic port.

Port 5 is used for the peripheral multiplexing box and therefore, is not available for your use.

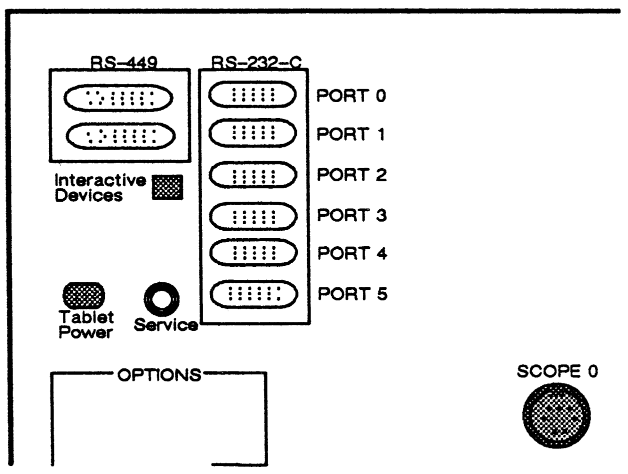


Figure 4. Port Configuration

- The Mass Memory section of the JCP card has one megabyte of memory with the option of a second megabyte available.
- The interface section of the JCP provides a location for the optional IBM 3278 interface. This option allows the PS 390 to communicate with an IBM 3274 control unit over a 56KB line. It is functionally equivalent to the PS 300 IBM 3278 GPIO card. Separate GPIO cards are available for high-speed communication interfaces other than IBM 3278.

2.2 Operating Specifications

Operating specifications for the PS 390 are as follows.

Grounding – The PS 390 scope should share a common ground with the control unit

Power Requirements – 115V Single Phase ±10% 47–63 Hz, 12 amp (max)
220V Single Phase: 7 amps (max) for the control unit

The following limitations are placed on AC power disturbances:

- A maximum of ±10% of nominal power for .1 seconds occurring no more than once every 10 seconds.
- Maximum harmonic content of 5% rms, no more than 3% rms for any single harmonic.
- Maximum impulse of 300V with rise time of .1 microseconds or slower, lasting no longer than 10 microseconds total duration.

Power Consumption – 1380 watts maximum

Heat Dissipation – 4710 BTUs/hour maximum

Operating Temperature – 65° to 80°F (18° to 27°C)

Relative Humidity – 20% to 80%

2.3 Multiplexing Box and Peripheral Connections

Peripherals for the PS 390 are connected to a multiplexing box contained in a three-inch pedestal that supports the raster scope. Mux boxes for either set of peripherals supported by PS 390 have the same operating instructions noted here. All peripheral connections for the mouse, function buttons, control dials, keyboard and tablet are clearly marked on the front panel of the mux box. Figure 5 shows the front view of the mux box.

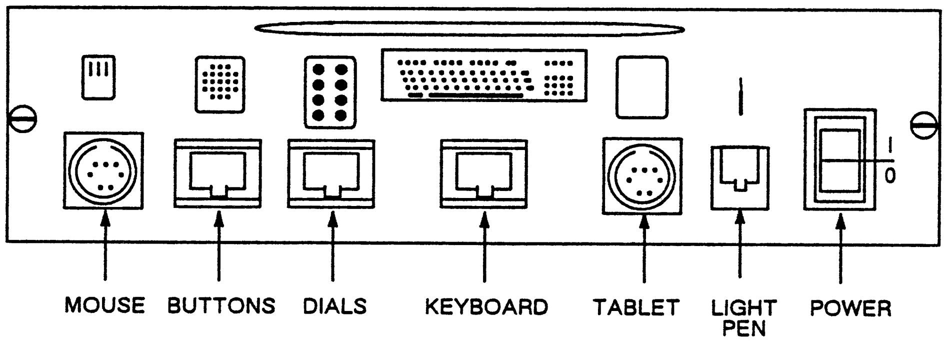


Figure 5. Front View of Multiplexing Box

The back panel of the multiplexing box has an RS232-C connection, three external power connections and two BNC connections marked LPICK and TPSW. The BNC connections are reserved for future use. All cables and connections are clearly marked. To maintain EMI integrity, the screws on the RS232-C shielded cable must be tightly turned on the connection. The rear panel of the mux box is shown in Figure 6.

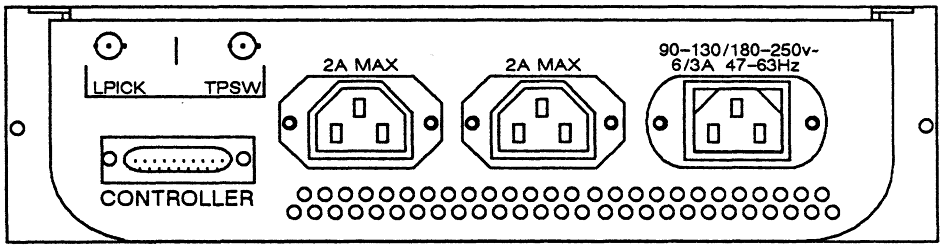


Figure 6. Rear Connections of Mux Box

Documentation on the new-style peripherals and multiplexing boxes is included as a separate manual with this release. Documentation for previous peripherals is contained in Volumes 1 and 5 of the *PS 300 Document Set.*

3. RUNTIME MODIFICATIONS AND NEW FEATURES

PS 390 runtime firmware supports new PS 390 functionality and existing PS 300 functionality. Assume that all functionality described in the *PS 300 Document Set* is correct and applicable to the PS 390 unless specifically noted as different in this and following sections.

As previously mentioned, the primary concern in developing the PS 390 runtime was maintaining user software compatibility with previous systems. Some incompatibilities exist because the reduced-size control unit does not support ports 0 and 2 for use with the DMR-11 interface or multi-user systems.

3.1 New PS 390 Function

A new initial function instance, PS390ENV, is provided. This function sets up display background color, and selects cursor and cursor color.

Input <1> is a trigger which accepts any data type to cause the function to run.

Input <2> is a constant which accepts a 3D vector (hue, saturation and intensity) to specify background color. The default background color is 0,0,0 (black). Saturation and Intensity must be in range of [0,1], otherwise an error message will be generated. Hue is in the range of [0,360]. For any value specified outside this range, multiples of 360 are added or subtracted to bring it into this range.

Input <3> is a constant which accepts an integer in the range [0,7] to specify the cursor color where

0 =	black
1 =	blue
2 =	green
3 =	cyan
4 =	red
5 =	magenta
6 =	yellow
7 =	white (default)

Any value outside this range generates an error.

Input <4> is a constant which accepts an integer to select the cursor.

0 = update rate cursor (default)
1 = system-defined refresh cursor

Input <5> accepts an integer to specify the video timing format, which is output from the video connection on the back of the PS 390 control unit.

0 = 1024 x 864 non-interlaced (default required by the PS 390 display)
2 = 1024 x 864 interlaced
3 = 640 x 484 interlaced (RS-170)

NOTE

When specifying the system-defined refresh rate cursor, you should leave the initial viewports HVP1$ and GVP0$ unchanged in order to have the (hardware) cursor work with picking.

3.2 Dynamic Viewport Considerations

Although the raster screen contains 1024 by 1024 addressable pixels, the actual displayable area on the raster screen is a rectangle, with pixel addresses going from 0 to 1023 in X and 0 to 863 in Y, where the physical pixel address 0,0 is in the lower left corner. A PS 300 viewport which spans (-1,1) in both vertical and horizontal directions maps onto the full 1024 x 1024 screen so that a rectangular portion along the lower edge of the viewport is not displayed. To avoid this situation, all viewports in the display structure are initially concatenated with a default viewport in the top display structure which maps to a square of 864 x 864.

The command

```
VPF1$ := Viewport Horizontal = -0.825:0.825 Vertical = -0.65:1 Intensity = 0:1
Then HVP1$;
```

in the boot-time configuration file accomplishes this.

If you want to override the default and use the entire displayable rectangular screen area, the following command can be entered:

```
Configure A;
VPF1$ := Viewport Horizontal = -1:1 Vertical = -0.65:1 Intensity = 0:1
```

```
Then HVP1$;
Finish Configuration;
```

This will cause all the subsequent VIEWPORT commands in the structure to be concatenated with this rectangular viewport. In doing so, however, your data must account for the non-square viewport.

To re-establish the default viewport, use either the commands

```
Configure A;
VPF1$ := Viewport Horizontal = -0.825:0.825 Vertical = -0.65:1 Intensity = 0:1
Then HVP1$;
Finish Configuration;
```

or

```
Screensave := F:Screensave;
```

Note that the initialize command does not restore the original viewport. Also, note that you cannot override the default viewport with the LOAD VIEWPORT command.

3.3 "Soft Labels" Function Network

Included on the distribution tape in the PS 390 Subdirectory is the "soft labels" ASCII file, which sets up a structure and network to use a normally unused portion of the screen to display function key and dial labels. This file can be incorporated in your SITE.DAT file if you have the new peripherals without LED labels. The labels appear on the left-hand side of the screen, with the square, default, graphics viewport shifted fully to the right. The displayed labels provide visual feedback to the user, but they are not pickable.

The structure and network requires no application software changes, except that the label, **flabel10**, no longer exists. (This is the 96 character label that goes across the entire LED area of the standard E&S keyboard with LEDs.) Figure 7 shows this soft labels area as it appears on the screen.

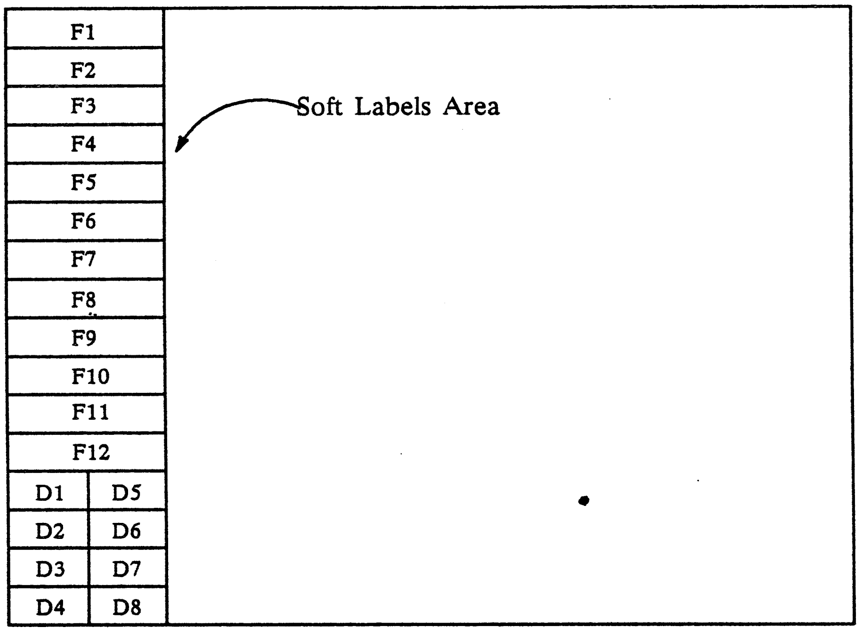


Figure 7. Screen Layout for the PS 390 with Soft Labels

3.4 Multiple GPIO Interfaces

The PS 390 runtime firmware supports up to two GPIO interfaces of differing types as well as asynchronous communications installed in the same system. The default configuration is asynchronous, but you have the ability to change your default to configure any interface when the system is booted. This is explained in the following section.

It is also possible to change the configuration without rebooting the PS 390 because the runtime determines which of the interfaces are in the system and initializes them all. This is achieved through runtime identification of up to two GPIOs at the first two addresses assigned to GPIO interface cards. (Refer to Send 'UNIBUS' command in 3.4.1 for an example of how to do this.) However, there are some limitations to the use of multiple GPIOs. First, there cannot be two of the same type GPIOs in the same system. Second, if the IBM 3278 option is included, then only one additional GPIO may be added. The 3278 GPIO running under previous PS 300 systems is not supported under PS 390. Table 1 shows the possible GPIO combinations.

Table 1. Possible GPIO Combinations

1st GPIO	2nd GPIO
IBM 3278 (enabled on JCP)	IBM 5080
	Parallel
	Ethernet/DECNET
IBM 5080	Parallel
	Ethernet/DECNET
Parallel	IBM 5080
	Ethernet/DECNET
Ethernet/DECNET	IBM 5080
	Parallel

3.4.1 Interface Configuration Files

The PS 390 runtime is distributed on two diskettes and contains more files than previous PS 300 runtime diskettes. This is to allow for the many different combinations of interfaces possible with the multiple GPIO operation.

When the PS 390 is booted, the system attempts to read the file, **INTFCFG.DAT**. If this file is not found, the system will boot with the default interface of Asynchronous, and display the message **INTFCFG.DAT NOT FOUND**. To boot with a default interface in addition to Asynchronous, the appropriate interface file must be renamed to INTFCFG.DAT. This can be done using the Diagnostic Disk Utility program described in Volume 5, Section 10 of the *Document Set*. For example,

```
Rename  ETHERNET.DAT  INTFCFG.DAT
```

would rename the default interface to Ethernet so that, at boot time, the communications interface protocol for Ethernet would be configured.

The following is a list of the file names on the diskette and which interface each file sets up.

ASYNC.DAT	Asynchronous communications
IBM3278.DAT	IBM 3278 communications
IBM5080.DAT	IBM 5080 communications
UNIBUS.DAT	Parallel interface communications
ETHERNET.DAT	Ethernet communications (for Ethernet or Decnet)

If your system hardware supports two interfaces, you can change the interface during a session without rebooting by sending the name of the interface file to input <1> of RDCFG$. For example, the following command,

```
Send 'UNIBUS' to <1>RDCFG$;
```

would change the communications protocol to the UNIBUS Parallel interface to allow parallel communications.

Table 2 shows the files contained on the PS 390 diskettes which are needed for a particular interface.

Table 2. Required Interface Files

PS 390 File Name	*Async*	*3278*	*5080*	*Unibus*	*Ethernet*
mmdd390J.EXS	✓	✓	✓	✓	✓
ACPCODE2.DAT	✓	✓	✓	✓	✓
ASYNC.DAT	✓				
CHARFONT.DAT	✓	✓	✓	✓	✓
CIRCLE.DAT			✓		
CONFIG.DAT	✓	✓	✓	✓	✓
DINTCODE.DAT					✓
EINTCODE.DAT					✓
ETHERNET.DAT					✓
FCNDICTY.DAT	✓	✓	✓	✓	✓
FCNTABLE.DAT	✓	✓	✓	✓	✓
FONT5080.DAT			✓		
GPIOCODE.DAT		✓			
HMSCODE.DAT	✓	✓	✓	✓	✓
HMSCOL.DAT	✓	✓	✓	✓	✓
HMSVEC.DAT	✓	✓	✓	✓	✓
IBM3278.DAT		✓			
IBM5080.DAT			✓		
IBMASCII.DAT	✓	✓	✓	✓	✓
IBMFONT.DAT	✓	✓	✓	✓	✓
IBMKEYBD.DAT	✓	✓	✓	✓	✓
INITACP.DAT	✓	✓	✓	✓	✓
INITGPIO.DAT		✓			
LINLUT.DAT	✓	✓	✓	✓	✓
LUT.DAT	✓	✓	✓	✓	✓
MSGLIST.DAT	✓	✓	✓	✓	✓
OVERLAY2.DAT	✓	✓	✓	✓	✓
PARSECODE.DAT	✓	✓	✓	✓	✓
PARSDICT.DAT	✓	✓	✓	✓	✓
PINTCODE.DAT				✓	
SINE.DAT			✓		
THULE.DAT	✓	✓	✓	✓	✓
UNIBUS.DAT				✓	

All of the interface files assume that the keyboard used is a VT100-style keyboard. A FALSE is sent to the keyboard handler (either IBMKBD or KBHANDLER) at the end of the file. To use the IBM-style keyboard, the command in the interface file must be changed to send TRUE to the keyboard handler. For example,

```
Send True to <2>Kbhandler;
```

would accomplish this.

This also means that full VT100 support is provided with the IBM-style keyboard.

(Please note that the IBM keyboard is not supported with the initial release of the PS 390.)

3.4.2 Ethernet/DECNET Interface

The GPIO interface hardware for Ethernet and DECNET is the same. The only difference is the microcode that is loaded into the GPIO. Therefore, both microcode files are distributed on each disk. The runtime attempts to load a file named EINTCODE.DAT. Ethernet is the default on the disk. The file for the DECNET interface is DINTCODE.DAT. If your system supports a DECNET interface, you must rename DINTCODE.DAT to EINTCODE.DAT to load the DECNET microcode into the GPIO. This can be accomplished by using the Diagnostic Utility program.

NOTE

As documented in the *Customer Installation and User Manual PS 300 Ethernet Interface*, you must send the assigned Ethernet address to the PS 300. The command to do this for the PS 390 is

```
Send 'address' to <1>ei_o1$;
```

Please refer to that manual for instructions on doing this.

3.5 Crash Dump File

Another of the new features of the runtime is the writing of a Crash Dump file to the diskette in drive 0 when a system crash occurs. This file is always named Crash.dat;1 and occupies only 1 block on the diskette.

If the file already exists it will be overwritten by the new crash information. If the file doesn't exist, it will be created. If there is insufficient room on the disk for the file, no crash dump file will be written.

The file consists of the 8 Data, the 8 Address registers, system version, system type, program counter, error type, error number, 59 32-bit stack entries, and the 68000 status register. Figure 8 shows the structure of the data in the crash file. Appendix A gives more information on some of these values.

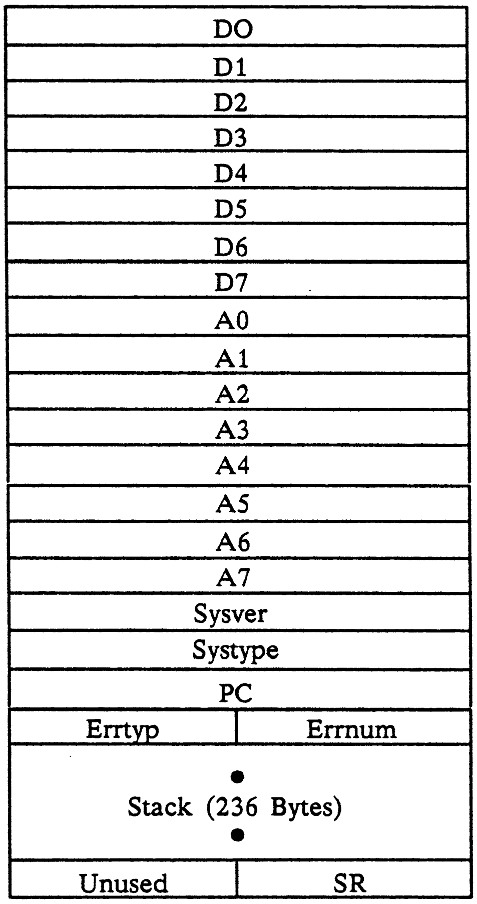


Figure 8. Data in Crash File

Appendix A also gives an example of a host PASCAL program that reads back the Crash.dat file from a PS 300. This program will read the Crash.dat file from a PS 300 and display the information in a format similar to the debug port on the PS 300. This information can be helpful in determining the cause of a crash.

The READDISK function has an added constant input <2> which accepts a boolean. If there is a true on input <2> after the file specified on input <1> is read, the file is deleted.

One possible use of this function is that an application program on the host could read and maintain crash file information. For example, a host program could have a start up procedure that checks to see if a crash file exists and then logs it in a host file. By reading and then immediately deleting this file, the program prevents the logging of crash files that were already recorded. The existence of a crash file would indicate that a crash had occurred since the last time the host program was run.

3.6 Additions to F:PICK

The PS 390 pick function, F:PICK, has three additional inputs. Input <4> is a real number between 0 and 1 that defines the pick window half size for the ACP pass of the pick . This is different from the size set by the SET_PICKing_LOCation operation node. The Line Generator or the Frame Buffer uses the operation node to determine if a pick has occurred, while the ACP uses input <4> to do the actual pick pass on the data.

Input <5> is an integer specifying pick pass retries. Since it is possible that the ACP will not find the picked data during a pick pass, input <5> indicates the number of times to add the window half-size increment on input <6> and try another pick pass.

Input <6> is a real number between 0 and 1 which specifies the amount to increase the pick window half size on each retry of the pick pass.

The defaults for each input are:

Input <4>	6.8359E-3
Input <5>	4
Input <6>	6.8359E-3

3.7 UWF Runtime Code Modification

The stack allocation scheme for User-written functions (UWF) has been changed. The UWF stack is now allocated when functions are downloaded rather than when they execute. As each function is processed by the SREC_GATHER function, the stack size requested is checked against the size of the currently allocated stack. If the requested stack size is the same or smaller, no action is taken. If the requested stack size is larger than the currently allocated stack, the current stack is disposed and a new one is allocated. All UWFs therefore use the same stack area. This allocation scheme eliminates the time previously required to allocate and deallocate a stack block on each execution of a UWF. Note that when UWFs are used, the one with the largest stack request should be loaded first.

4. PS 390 EXCEPTIONS

PS 390 functionality is the same as described in existing PS 300 documentation with the exception of the following.

There is no support for Port 0 and Port 2 of the control unit, DMR-11 interface, multi-user, or any scope other than scope 0.

The default viewport of the PS 390 is 864x864 centered on the raster display. You can change this by using the commands described in Section 3.2, but you cannot modify the this default viewport using the Load Viewport command.

Local hardcopy is not supported. Host hardcopy is available through the Writeback feature included with this release.

Reading back of pixel data from the PS 390 to the host is not supported nor is the loading of user-defined color lookup tables or the display of anti-aliased objects in overlay mode. This functionality is planned for future releases.

The lightpen is not supported.

There are no vector-normalized vectors; all ASCII and GSR vector list commands which do not specify block-normalized vectors will create 32-bit block-normalized vectors internally in the PS 390. (No modifications to ASCII commands or GSR routines are required.)

There is no "per vector" intensity specification available.

There is no color blending (color by vector) available.

Zero length vectors cannot be picked.

There is no allowance for the display of transformed data (data output by F:XFORMDATA). However, a limited form of access to the data generated by the F:XFORMDATA function has been provided to allow certain user-written functions (and the CPK modeling firmware) to perform properly. Please note the following restrictions on the use of transformed data on the PS 390:

- F:XFORMDATA outputs a non-displayable data type (vector-normalized vector list).
- A single-precision vector list is generated by F:XFORMDATA.
- Only three-dimensional data can be transformed.

F:XFORMDATA can still be connected to F:LIST to enable the host to read the transformed data retrieved from the PS 390.

Existing PS 300 applications that create nodes with functionality not yet supported by the PS 390 will be treated as no operation nodes.

APPENDIX A

Crash Dump Information

The System Version is a number generated indicating the date the runtime was created. For example, a value of 111486 means the system was created on Nov. 14, 1986.

The system type is a three digit number indicating the type of system that is being used. The first digit on the left is a 1 for GCP, 2 is reserved, 3 for JCP. The second digit, 1, is reserved, 2 is for 320, 3 is for 330, 4 is for 340, 5 is for 350 and 6 is for 350/340. The last digit is 0 for Async or any JCP, 1 for IBM 3278, 2 for Parallel, 3 for IBM 3250/5080 and 4 for Ethernet/DECNET. Digit 3 will always be 0 for JCP systems. For example a value of 350 indicates JCP 350.

Error Types/Error Numbers

There are three crash error types in the PS 300. Each type has a set of error numbers associated with the type. The three types are:

1. System Errors
2. Traps
3. Exceptions

The following is the list of errors for each type.

Type 1 - System Errors

1	Track number out of range
2	Disk drive not ready
3	Disk remains busy after a seek
4	Block number out of range

6	Lost data during read
7	Record not found during read
8	Data CRC error during read
9	ID CRC error during read
B	Lost data during write
C	Record not found during write
D	Data CRC error during write
E	ID CRC error during write
F	Write fault
10	Disk is write protected
11	Lost data during format
12	Write fault during format
14	Disk drive number out of range
15	Seek error
16	Drive not ready during read
17	Drive not ready during write
18	Disk not at track Ø after restore command
19	Disk busy after restore command
1A	Track number out of range during format
1B	Drive not ready during format
1C	Disk write protected during format
1D	Time out during read
1E	Time out during write
1F	Time out during format
64	Wait maybe called with nil argument
65	Wait maybe called with a non-function
66	Wait maybe, already a function waiting
67	Wait maybe, parameter function waiting elsewhere
68	Q ship to an unrecognized Namedentity
69	Msgcopy, Message type shouldn't be copied
6A	Msgcopy, Msg type Has structure, unknown to Msgcopy
6B	Send, 'Me' = nil
6C	Send, 'Me' not a function instance
6D	Send, No such output port for this function
6E	Rem_conn/Add_conn, A1 = nil
6F	Add_conn, A2 = nil
70	Findqueue, Named item = nil
71	Findqueue, illegal queue number (queue no. < Ø or queue no. > no. of inputs for function)
72	Allinpwait, Nmin > Nmax
73	Allinpwait, Nmin < 1
74	Tmessage, Waiting and n = Ø
75	Cmessage, Waiting and n = Ø
76	Lookmessage, Waiting and n = Ø
77	Allinputs, Nmin > Nmax

78	Allinputs, Nmin < 1
79	Fcnnotwait, Me = nil
7A	Findqueue, found a nil queue!
7B	Waitnextinput, n = Ø
7C	Anyoutputs, Me = nil
7D	Anyoutputs, illegal outset number
7E	Anyoutputs, no outset where there should be
7F	Fdispatch, function failed to re-queue after running
80	Text_text, B1 < Ø
81	Char_text, b < Ø
85	Error during disk read
8D	Initial structure not correct
8E	AnnounceUpdate List tail = nil;head < > nil
8F	FormatUpdate Somebody's sleeping in my bed
90	FormatUpdate Ready Head not nil but Tail is
91	Bad code file -- illegal Op
92	ByteIndex Invalid Acpdata type
93	FormatUpdate, PASCAL Head not nil but Tail is
94	Vec_size, Invalid Acpdata type
95	KillUpdate, Updfetch was < Ø
96	KillUpdate, Some one was sleeping in my bed
97	Vec_bias, Invalid Acpdata type
99	CntCapacity, Invalid Acpdata type
9C	Unknown brand of Namedentity
9D	Hasstructure knows something I don't
9E	Amuhead not a Qalphapair
A1	AppendVector, Invalid Acpdata type
A3	Nomemsched, Bad .Status for a fcn
A9	Bad update list on ACP time-out
AA	ACP Timeout during initialization
AB	Crashprepare, Name CRASH$ has not been defined
AC	DecUpdsync, C_header ^ .Updsync < Ø
AD	FormatUpdate, Someone waiting in C_header ^ .Updswait already
AF	Someone else waiting in C_header ^ .Killer already
BO	Non-nil Qwait of a dying function
B3	Microcode won't fit into ACP
B4	Implementation limit on delta waits (2**31)
B8	detected internal inconsistency
B9	detected error (passed a bad parameter)
BA	diskette's parsecode table inconsistent with parser
BD	Bad boundary on binary data xfer
BF	default Devsts contains errors
C0	Inwait, f is already waiting or not a function
C1	Outwait, f is already waiting or not a function
C2	ECO Level of GCP does not support 56K Baud Line

C3	Port 1 Configuration is invalid for 56K Baud Line Support
C9	User generic function stack overflow
CA	Ug_run_cnt has become negative
CB	User generic function has bad alpha (on private queue)
CC	Bad format of MSGLIST .DAT detected
CD	MSGLIST (or code using it) has probably been corrupted
CF	Apparent datastructure incompatibility
D0	Bad MemOKindex detected
D1	routine passed bad parm (e.g., a nil ptr)
D2	Lines to IBM system not active
D3	Floppy disk file INITGPIO.DAT; not found or unable to read
D4	Floppy disk file GPIOCODE.DAT; not found or unable to read
D5	Floppy disk file IBMFONT.DAT; not found or unable to read
D6	Floppy disk file IBMKEYBD.DAT; not found or unable to read
D7	Floppy disk file IBMASCII.DAT; not found or unable to read
D8	IBM GPIO timeout
D9	No. of minimum inputs is negative
DA	No. of maximum inputs < No. of minimum inputs
DB	No. of maximum inputs > # inputs for function
DC	Sendlist detected a bad list
DE	Sendmess: message to be sent is NIL
DF	Caller did not have a lock set already
E0	Curfcn in improper state to call Getinputs
E1	Cleanin, Curfcn in improper state to call Cleaninp (e.g., have you first called Getinputs?)
E2	Somebody remembered a forgotten non-fcninstance
E5	Alpha not already locked by caller
E6	Confusion in discarding bad message
E7	Lock not already set by caller
E8	Probable multiple master GCPs
E9	RemOne, Curfcn does not have that many inputs
EA	RemOne, Message to be deleted and message pointed to by Curinputs is not the same
EB	Lock not already set in Gatheraupdate call
ED	Get2locks detected lock already set
EE	Error in semantic routine for polygon vertex
EF	Destination Alpha was not already locked
F0	Parent not already locked in add/remove from set
F1	Child not already locked in add to set
F3	Alpha not already locked in Gpseudoaupdate
F6	Confusion about locks or decausages
F7	Unknown tap reason
F8	Unanticipated state at which to see shoulder tap
F9	Illegal number of inputs
FC	No existing DCB found for this user

FD	Timeout, Message on input 1 disappeared before fcn could get it
FE	Error while initializing disk drive
FF	Error while reading disk header
100	Error while reading disk directory
101	THULE.DAT not found on disk
102	Error while reading THULE.DAT
103	Curfcn was not active at entry
104	Viewport not in structure
105	Real_simple, number of digits requested out of range (n < 1 or n > 9)
106	Getnextone, illegal queue specified
107	Getnextone, msg on head of queue and specified by Curinput do not agree
108	Getnextone, no message on queue, but Curinput < > NIL
109	ContBlock, nil block
10A	Timeout when waiting for all on-line GCPs
10B	Rehash only works first time, only time now.
10C	No processor has right to issue this tap
10D	GetVector, Not an Acpdata block
10E	GetVector, Not a vector Acpdata block
10F	Invalid qpacket received
110	Tolerance on FCnearzero is absurd
111	set construct of father has no dummy control block
112	function code has to be of type CI to have elements included and removed
113	ShadeEnviron node encountered in non PS 340

Type 2 - Traps

0	No mass memory on line, or too little to come up
1	More OKINTs than NOINTs or > 128 NOINTs
2	Free storage block size bad (on request or in free list)
3	Attempt to Activate a non-function (or nil) or bad software detected during startup (most commonly, incompatible datastru.sa detected but perhaps invalid startup routine sequencing (if someone has been mucking around with it))
4	NEW call failed to find memory, within NOMEMSCHED
5	Attempt to queue where a function is already waiting
6	Systemerror(n)
7	Badfcode(Fcn)
8	Mass Memory Error Interrupt
9	Utility Routine not included in this linked system
A	Probable multiple DISPOSE of the same block
B	Block exponent not big enough
C	Attempt to divide with a divisor which is too small in FixLongDivide(twice the dividend must be less than the divisor)
D	(Used by Motorola PASCAL)

Type 3 - Exceptions

0	Reset: Initial SSP
1	Reset: Initial PC
2	Bus Error (i.e. attempt to address nonexistent location in memory)
3	Address Error (i.e. attempt to access memory incorrectly, for example an instruction not starting on a word boundary).
4	Illegal instruction
5	Zero Divide
6	CHK Instruction
7	TRAPV Instruction
8	Privilege violation
9	Trace
10	Line 1010 Emulator
11	Line 1111 Emulator
24	Spurious interrupt

Crash Dump Program

Following is an example of a Pascal host program that writes the information from the PS 300 crash file into a host file.

```
PROGRAM CRASH (Input,Output,Outfile);

CONST
        %INCLUDE 'PROCONST.PAS/NOLIST'

TYPE
        %INCLUDE 'PROTYPES.PAS/NOLIST'

        cheat_4 = RECORD

                CASE Boolean OF
                        TRUE : (i : Integer);
                        FALSE : (c : Array[1..4] OF CHAR)

                END;

        cheat_2 = RECORD

                CASE Boolean OF
                        TRUE : (i : [WORD] 0..1024);
                        FALSE : (c : Array[1..2] OF CHAR)

                END;

        Buffer = RECORD
                CASE Boolean OF
                        TRUE : (b : P_VaryBuftype);
                        FALSE : ({ Length of P_VaryBuftype is in Dummy}
                                Dummy           :[WORD] 0..1024;
                                Dreg            :Array[0..7] of Cheat_4;
                                Areg            :Array[0..7] of Cheat_4;
                                SVer            :Cheat_4;
                                Stype           :Cheat_4;
                                PC              : Cheat_4;
                                Errtyp          : Cheat_2;
                                Errnum          : Cheat_2;
                                Stack           : Array[1..59] of Cheat_4;
                                Not_Used        : Cheat_2;
                                SR              : Cheat_2)

            END;
```

```
 VAR
        Devtyp: Integer;
        Inbuff : P_VaryBuftype;
        OutBuff: Buffer;
        Found : BOOLEAN;
        Outfile : text;

%INCLUDE 'PROEXTRN.PAS/NOLIST'

%INCLUDE 'VAXERRHAN.PAS/NOLIST'

PROCEDURE Init_ps300;

        FUNCTIONAL DESCRIPTION:

            Initialize the comm link to the PS 300


        }


        VAR
            a, Modify : P_Varyingtype;

        BEGIN
        Write('Enter Type of Interface (1=Async, 2=Ethernet, 3=Parallel):');
        Readln( Devtyp );
        Write('Enter Device name :');
        Readln( a ); CASE Devtyp OF
            1 :
              Modify := 'LOGDEVNAM=' + a + '/PHYDEVTYP=ASYNC';
            2 :
              Modify := 'LOGDEVNAM=' + a + '/PHYDEVTYP=ETHERNET';
            3 :
              Modify := 'LOGDEVNAM=' + a + '/PHYDEVTYP=PARALLEL'
            OTHERWISE

            END;
        PAttach( Modify, PI_Error_handler)
        END;
```

```
PROCEDURE Trigger_read;

        FUNCTIONAL DESCRIPTION:

            Create instance of function network to retrieve CRASH.DAT file from disk. The
            network will convert the data block to six-bit format and break it into packets of 72
            bytes which will be put on host_message.


        }

        VAR

            a : CHAR;

PROCEDURE BREAKUP;
{ Code generated by Network Editor 1.08 }
{ This function network takes an incoming qpacket and breaks it }
{ into smaller packets to be sent over a terminal line since  }
{ most terminal handlers have some limit to the input length  }
{ BREAKUP }
BEGIN
{ Frame1: }
    PFnInstN ('Break_sync', 'SYNC', 2,
        PI_Error_handler);
    PFnInst ('Break_route', 'BROUTEC',
        PI_Error_handler);
    PFnInst ('Add_constant', 'CONSTANT',
        PI_Error_handler);
    PFnInst ('Break_add', 'ADDC',
        PI_Error_handler);
    PFnInst ('Breakup', 'TAKE_STRING',
        PI_Error_handler);
    PFnInst ('In_length', 'LENGTH_STRING',
        PI_Error_handler);
    PFnInst ('Len_compare', 'GTC',
        PI_Error_handler);
    PFnInst ('Route_string', 'BROUTE',
        PI_Error_handler);
    PFnInst ('Route_start', 'BROUTE',
        PI_Error_handler);
    PFnInst ('cvt', 'CVT8TO6',
        PI_Error_handler);
    PFnInst ('rd', 'READDISK',
        PI_Error_handler);
```

```
PFnInst ('prnt', 'PRINT',
    PI_Error_handler);
PFnInst ('Breakup_in3', 'CONSTANT',
    PI_Error_handler);
PConnect ('Break_sync', 1, 1, 'Breakup',
    PI_Error_handler);
PConnect ('Break_sync', 1, 2, 'Break_route',
    PI_Error_handler);
PConnect ('Break_sync', 2, 2, 'Breakup',
    PI_Error_handler);
PConnect ('Break_sync', 2, 2, 'Break_sync',
    PI_Error_handler);
PConnect ('Break_sync', 2, 2, 'Break_add',
    PI_Error_handler);
PConnect ('Break_route', 1, 1, 'Add_constant',
    PI_Error_handler);
PConnect ('Break_route', 1, 2, 'Route_string',
    PI_Error_handler);
PConnect ('Add_constant', 1, 1, 'Break_add',
    PI_Error_handler);
PConnect ('Break_add', 1, 2, 'Break_add',
    PI_Error_handler);
PConnect ('Break_add', 1, 2, 'Route_start',
    PI_Error_handler);
PConnect ('Break_add', 1, 1, 'Len_compare',
    PI_Error_handler);
PConnect ('Breakup', 1, 1, 'cvt',
    PI_Error_handler);
PConnect ('Breakup', 2, 1, 'Break_route',
    PI_Error_handler);
PConnect ('Breakup', 2, 1, 'Breakup_in3',
    PI_Error_handler);
PConnect ('In_length', 1, 2, 'Len_compare',
     PI_Error_handler);
PConnect ('Len_compare', 1, 1, 'Route_string',
    PI_Error_handler);
PConnect ('Len_compare', 1, 1, 'Route_start',
    PI_Error_handler);
PConnect ('Route_string', 2, 1, 'Breakup',
    PI_Error_handler);
PConnect ('Route_start', 2, 2, 'Breakup',
    PI_Error_handler);
PConnect ('cvt', 1, 1, 'host_message',
    PI_Error_handler);
PConnect ('rd', 1, 1, 'Break_sync',
            PI_Error_handler);
        PConnect ('rd', 1, 1, 'In_length',
            PI_Error_handler);
        PConnect ('rd', 2, 1, 'prnt',
            PI_Error_handler);
        PConnect ('prnt', 1, 1, 'host_message',
            PI_Error_handler);
        PConnect ('Breakup_in3', 1, 3, 'Breakup',
            PI_Error_handler);
        PSndStr(CHR(36), 2, 'cvt',
            PI_Error_handler);
        PSndFix (48, 3, 'Breakup',
            PI_Error_handler);
        PSndFix (48, 2, 'Breakup_in3',
            PI_Error_handler);
        PSndFix (48, 2, 'Add_constant',
            'PI_Error_handler);
        PSndFix (1, 2, 'Break_sync',
            PI_Error_handler);
        PPutPars('Set priority of prnt to 9; ',
            PI_Error_handler);
    END;

    BEGIN
    IF Devtyp = 1
    THEN
        Breakup
    ELSE
        BEGIN
        PFnInst ('rd', 'READDISK',
            PI_Error_handler);
        PFnInst ('prnt', 'PRINT',
            PI_Error_handler);
        PConnect ('rd', 2, 1, 'prnt',
            PI_Error_handler);
        PConnect ('prnt', 1, 1, 'host_message',
            PI_Error_handler);
        PConnect ('rd', 1, 1, 'host_message',
            PI_Error_handler);
        PPutPars('Set priority of prnt to 9; ',
            PI_Error_handler);
        END;
    Write(' Do you want to delete CRASH.DAT after reading?');
    Readln( a );
    IF (a = 'Y') OR (a = 'y')
```

```
THEN
    Psndbool( TRUE, 2, 'rd', PI_Error_handler)
ELSE
    Psndbool( FALSE, 2, 'rd', PI_Error_handler);
Psndstr( 'CRASH', 1, 'rd', PI_Error_handler)
PPurge( PI_Error_handler );
END;

PROCEDURE Get_data_block;

    {

    FUNCTIONAL DESCRIPTION:

            Read in data from PS 300, convert to 8 bit and put in buffer

    {
    VAR
        i,j,Temp : Integer;
        Done : BOOLEAN;

PROCEDURE Cvt_6_8
        (Inblock : P_VaryBuftype;
        VAR Outblock : P_VaryBuftype;
        Factor : Integer);

    VAR
        w : cheat_4;
        c_out,cycle_count,il,tc : INTEGER;
        First : BOOLEAN;

    BEGIN
    i := 1;
    First := TRUE;
    Cycle_count := 1;
    c_out := 4;
    WHILE i <= LENGTH(Inblock) DO
        BEGIN
        tc := ORD(Inblock[i]) - Factor;
         IF First
        THEN
            IF tc < 0
            THEN
                c_out := 4 + tc
            ELSE
            BEGIN
            First := FALSE;
            w.i := tc;
            cycle_count := SUCC(cycle_count)
            END



        ELSE
            BEGIN
            w.i := w.i * 64;
            w.i := w.i + tc;
            cycle_count := SUCC(cycle_count)
            END;
        IF cycle_count > 6
        THEN
            BEGIN
            FOR il := 1 TO c_out DO
                Outblock := Outblock + w.c[il];
            cycle_count := 1;
            First := TRUE
            END;

        i := SUCC(i);
        END;
    END;


BEGIN
Done := FALSE;
Found := TRUE;
WHILE NOT Done DO
    BEGIN
    Pgetwait( Inbuff, PI_Error_handler);
    IF Inbuff = '''TRUE '''
    THEN
        Done := TRUE
    ELSE
        IF Inbuff = '''FALSE'''
        THEN
            BEGIN
            Done := TRUE;
            Found := FALSE
            END
        ELSE
            IF Devtyp = 1
            THEN
                Cvt_6_8( Inbuff, Outbuff.b, 36)
            ELSE
                FOR i := 1 TO 80 DO
                    FOR j := 4 DOWNTO 1 DO
                        Outbuff.b := Outbuff.b + Inbuff[(i-1)*4 + j];
        END;

{ It is necessary to reverse Errnum with Errtyp }
{ and Not_Used with SR }
    IF Found
    THEN
        WITH Outbuff DO
            BEGIN
            Temp := Errnum.i;
            Errnum.i := Errtyp.i;
            Errtyp.i := Temp;
            Temp := Not_Used.i;
            Not_Used.i := SR.i;
            SR.i := Temp;
            END;


    END;
```

```
PROCEDURE Display_crash;

    {

    FUNCTIONAL DESCRIPTION:

        Display Crash info on terminal

    {
    PROCEDURE Dumpit;

        VAR
            SP,j,k,sloc,cloc : INTEGER;
             tc : CHAR;
            Sline : PACKED ARRAY [1..15,1..16] OF
                    CHAR;


        BEGIN
        Rewrite(Outfile);
        WITH Outbuff DO
        BEGIN
        Writeln(Outfile);
        Write(Outfile,' PC=',HEX( PC.i, 8, 8 ));
        Write(Outfile,' SR=',HEX( SR.i, 4, 4 ));
        Write(Outfile,' STYPE=',Stype.i:3);
        Write(Outfile,' SVER=',Sver.i:6);
        Write(Outfile,' ETYPE=',HEX( Errtyp.i, 4, 4));
        Write(Outfile,' ENUM=',HEX( Errnum.i, 4, 4));
        Writeln(Outfile); Write(Outfile,'D0-D7=');
        FOR j := 0 TO 7 DO
            Write(Outfile,' ',HEX( Dreg[j].i, 8, 8));
        Writeln(Outfile);
        Write(Outfile,'A0-A7=');
        FOR j := 0 TO 7 DO
            Write(Outfile,' ',HEX( Areg[j].i, 8, 8));
        Writeln(Outfile);
        Writeln(Outfile);
        Writeln(Outfile,'STACK='); SP := Areg[7].i + 14;
        FOR j := 1 TO 15 DO
            BEGIN
            Sloc := (j-1) * 4 + 1;
            Cloc := 4;
            FOR k := 1 TO 16 DO
                BEGIN
        IF sloc < 60
        THEN
            BEGIN
            tc:= Stack[sloc].c[cloc];
            IF tc > CHR(127)
            THEN
                tc:= CHR(ORD(tc) - 128);
            IF (tc < CHR(32)) OR
                (tc = CHR(127))
            THEN
                Sline[j,k] := '.'
            ELSE
                Sline[j,k] := tc
            END
        ELSE
            Sline[j,k] := '.';
        Cloc := Cloc - 1;
        IF Cloc = 0
        THEN
            BEGIN
            Cloc := 4;
            Sloc := Sloc + 1
            END;
        END;
    Write(Outfile,HEX( SP, 8, 8),' ');
    Sloc := (j-1) * 4 + 1;
    Cloc := 4;
    FOR k := 0 TO 15 DO
        BEGIN
        IF sloc < 60
        THEN
            Write(Outfile,' ',HEX( ORD(Stack[sloc].c[cloc]), 2, 2))
        ELSE
            Write(Outfile,' 00');
        Cloc := Cloc - 1;
        IF Cloc = 0
        THEN
            BEGIN
            Cloc := 4;
            Sloc := Sloc + 1
            END;
        END;
    Write(Outfile,'  ');
    FOR k := 1 TO 16 DO
        Write(Outfile,Sline[j,k]);
```

```
            Writeln(Outfile);
            SP := SP + 16
            END
        END
   END;


  BEGIN
  IF Found
  THEN
      Dumpit
  ELSE
      Writeln(' Crash file not found ')
  END;


BEGIN
Init_ps300;
Trigger_read;
Get_data_block;
Display_crash;
PDetach( PI_Error_handler);
END.
```

PART II

Change Pages And Previous Graphics Firmware Release Notes

A consolidation of Versions A1.V02 and A2.V01 of the *PS 300 Graphics Firmware Release Notes* are included in this package. Current customers should already have this information. Also included are change pages specific to PS 390 functionality.

The following commands and functionality have been added since the publication of the *Document Set*. The new commands have been formatted as supplement pages for the *PS 300 Command Summary*. The list below gives the new commands and a brief description.

Load Viewport	Loads a viewport and overrides the previous viewport (can not be used to modify default viewport).
Set Blinking ON/OFF	Creates blinking nodes to specify whether blinking is enabled in the specified structure.
Set Blink Rate	Specifies the blink rate.
Set Line_Texture	Specifies pattern for hardware texturing of displayed lines.
Writeback	Enables writeback for the data structure below the writeback node.
Rawblock	Allocates memory that can be directly managed by a user–written function, or the Parallel or Ethernet Interfaces.

DOCUMENTATION INFORMATION FOR ALL USERS

Important corrections to errors in the PS 300 Document Set are provided on the following pages. Please note these changes in your document set. New pages for previously undocumented functions are included here.

Several documents have been changed. The documents and the changes are summarized below. If you would like to have the newest version of any of these documents, please contact your E&S Account Executive.

- User-Written Functions: revised to correct errors in the document and provide templates with more complete instructions, as well as more information on writing various types of functions.

NOTE

A2.V02 - This manual has been completely revised and included in the PS 300 Advanved Programming manual that has been provided to you as a seperate document for the A2.V02 release.

- NETEDIT: revised to support the new version of NETEDIT.
- Introduction to Data Driven Programming Methodology: notes have been added to this document to clarify misleading information.
- PS 300 Application Notes: new Notes have been added.

Information for PS 300/IBM 3278 Interface Users

Enhancements and New Features in the PS300/IBM 3278 Firmware

- The PS 300/IBM 3278 Terminal Emulator Setup mode now includes keys that will inhibit the display of the cursor, the PS 300 indicator characters, and the host indicator characters. Inhibition of these screen characters is accessed by entering Setup mode, (ALT/GRAPH or ALT/SETUP on the IBM 3278-style keyboard, SETUP on the VT100-style keyboards) and toggling the appropriate keys.

  Once in Setup (shown by the display of the PS 300 indicator character 'S' on the bottom line of the screen), the following new Setup features are available:

FUNCTION KEY	FEATURE
F6	Toggles the display of the PS 300 characters. Default is the display of the characters.
F7	Toggles the display of the host indicator characters. Default is the display of the characters.
F8	Toggles the display of the cursor. Default is display of the cursor.

  Function keys F9 and F10 are used in conjunction with the PS 300/IBM 3250 Interface. Information on the use of these keys is available in the PS 300/IBM 3250 Interface User's Manual.

  The adjustments made in Setup can be entered as PS 300 commands in the SITE.DAT file to set the appropriate characteristics at boot time.

  The list below shows the characters that should be entered into the SITE.DAT file for each new feature.

  For VT100-style keyboards, the appropriate character(s) must be inserted between a '↑Vo ↑Vo' header and trailer sequence. ↑Vo is a CTRL V lowercase "o" sequence:

FEATURE	CHARACTERS TO BE ENTERED INTO SITE.DAT
Set/Reset Local Indicators	SEND '↑Vo↑Vf↑Vo' TO <1>IBMKBD1;
Set/Reset Host Indicators	SEND '↑Vo↑Vg↑Vo' TO <1>IBMKBD1;
Set/Reset Cursor	SEND '↑Vo↑Vh↑Vo' TO <1>IBMKBD1;
Set 3250 Mode	SEND '↑Vo↑Vi↑Vo' TO <1>IBMKBD1;
Set PS300 Mode	SEND '↑Vo↑Vj↑Vo' TO <1>IBMKBD1;

For IBM-style keyboards, the appropriate characters must be inserted between a CHAR(130)&CHAR(n)&CHAR(130) sequence, where &CHAR(n) is the character sequence(s) for the feature:

FEATURE	CHARACTERS TO BE ENTERED INTO SITE.DAT
Set/Reset Local Indicators	SEND CHAR(130)&CHAR(150)&CHAR(130) TO <1>IBMKBD1;
Set/Reset Host Indicators	SEND CHAR(130)&CHAR(151)&CHAR(130) TO <1>IBMKBD1;
Set/Reset Cursor	SEND CHAR(130)&CHAR(152)&CHAR(130) TO <1>IBMKBD1;
Set 3250 Mode	SEND CHAR(130)&CHAR(153)&CHAR(130) TO <1>IBMKBD1;
Set PS300 Mode	SEND CHAR(130)&CHAR(154)&CHAR(130) TO <1>IBMKBD1;

- A kit containing keycap replacements for the 3278-style keyboard accompanies this release. The configuration of the keyboard has changed with the A1.V02 Firmware to support additional keys required by some IBM applications. The keyboard reconfiguration and keycap replacements are as follows:

  1. The keys designated for use by IBM applications are the old GRAPH and TERM keys on the left-hand keypad of the keyboard. These keycaps will be replaced by blank keycaps and have no PS 300 application.

  2. The old SETUP and TEST/NORM keys on the left-hand keypad will become dual-purpose keys. The new keycap for the SETUP key will read GRAPH on the top and SETUP on the front of the key. To access Setup mode, the key must be pressed in conjunction with the ALT key on the keyboard.

  The new keycap for the TEST/NORM key will read TERM on the top and TEST/NORM on the front. The terminal display will be toggled on and off by pressing the key. To access TEST/NORM, the key must be pressed in conjunction with the ALT key on the keyboard.

  The keycap exchange will be made by the user. Additional instructions for changing the keycaps are included in the kit.

- Two system functions (F:IBM_KEYBOARD and F:IBM_SETUP) have been modified to support the new PS 300/IBM 3250 Interface. The modifications made to these functions are shown on the System Functions change pages. These pages may be inserted into Volume 5 of the PS 300 Document Set.

The following section contains the NETEDIT Release Notes.

Version A1.V02 - March 1985

NETEDIT V1.08 RELEASE NOTES

A revised version of the NETEDIT programming tool is provided on the magnetic tape distributed with the A2.V02 PS 390 Firmware. A description of changes follows. If you wish a new version of the NETEDIT User's Guide, contact your E&S Account Executive to order the updated documentation.

FORTRAN/Pascal GSR Code Conversion

There are now options to produce FORTRAN or Pascal code, as well as the usual PS 300 ASCII commands, available under CONVERT NETWORK. Selecting these options produces a subroutine or procedure which can be compiled and linked with a user-supplied main program and the appropriate GSR library.

The Pascal code is compatible with VAX/VMS Pascal V2; the FORTRAN code is compatible with VAX/VMS FORTRAN-77.

The menu items ASCII OUTPUT, FORTRAN GSR, and Pascal GSR cause the corresponding type of code to be generated. The other menu items toggle various options on and off; you should set these before you select the item to produce the code.

You must take special care to see that the code for all macros referenced in your network have been converted to the same form (i.e., ASCII, FORTRAN, or Pascal) as the code to be produced for the rest of the network. For example, when you are generating Pascal code you cannot reference an ASCII macro. NETEDIT will give a warning message if you attempt to do this.

The following discussion of how to compile and link the generated code with your program assumes familiarity with the GSRs.

The generated code is output to a file with the same name as the network, with an extension of .PAS for Pascal, and .FOR for FORTRAN. The code is in the form of a single subroutine or procedure with the same name as the network; this routine takes no arguments.

Your program must perform the calls to attach and detach the PS 300 (PAttach/PDetach for Pascal, PATTCH/PDTACH for FORTRAN). You must also supply an error handling routine, as described in the GSR documentation.

For FORTRAN, the error handler must be named ERR. The output file produced by NETEDIT may be compiled independently, or included in a file containing other FORTRAN subprograms. You must then link it with your main program, the error handler, and the FORTRAN GSR library.

For Pascal, the error handler must be named PI_Error_Handler. The suggested method for compilation is to include the file containing the generated code in your main program file, using the %include directive. Your program must also include the declarations in PROCONST.PAS, PROTYPES.PAS, and PROEXTRN.PAS. After compiling the program, you must link it with the Pascal GSR library.

Literal PS 300 Commands Can Be Included In Network

Specially flagged labels can be used to insert random PS 300 commands in a network. Floating comments which start with \+\ or \-\ indicate commands to be inserted before or after the other code for the frame, respectively. These commands are always written to the output file during code conversion, regardless of the SUPPRESS COMMENTS setting.

The statements can be ordered by including a priority number in the flag. For example, statements prefixed with \-1\ are guaranteed to be sent before statements prefixed with \-2\. This is useful for sending an ordered sequence of constants to the same input of a function, for example.

Typically, commands that should be inserted before the other code for a frame are initialize commands or display structure definitions. Commands that should be specified to go at the end of the code for the frame are SETUP CNESS commands, and SEND statements. NETEDIT does not perform any syntax or validity checking on the commands.

Names of functions, variables, and display structures that are referenced in these commands may be prefixed with \F\ and/or \M\ to indicate that the appropriate frame and/or macro prefix should be substituted during code conversion.

Version A1.V02 - March 1985

NETEDIT Now Uses GSRs

NETEDIT has been changed to use the GSR library internally. This should result in some increase in performance for those using high-speed lines. The device type may be specified using the @AttachTo option in the parameter file. The default value, for the RS-232 async line, is:

```
@ATTACHTO logdevnam=tt:/phydevtyp=async
```

See the GSR user's manuals for more information on how to specify this parameter.

If the Pascal GSR library is not available, a library of procedures with the same calls as the GSR routines, but which send the equivalent ASCII commands to the parser, is provided.

Support Network Uses UWFs

Some parts of the support network have been replaced by user-written functions. No new functionality has been added, but users may notice some improvement in performance. NetEdit V1.08 will not work with PS 300 firmware that does not support UWFs (i.e., pre-A1 firmware).

Improved Handling of Arcs

Users should see faster response when adding arcs as a result of changes to the host program and the support network. Adjacent colinear segments are now combined when the arc is processed. In addition, better ways for handling arcs when the items they are attached to have been moved should cut down on the need to manually reroute arcs.

Improved Text Editing Facilities

NETEDIT now uses an improved line editing function for text entry. This function behaves like a one-line screen editor, similar to EMACS in its use of control characters for editing effects. If you are editing an existing piece of text, you do not have to retype the entire line just to make a minor change, as the buffer is initialized to contain the previous contents of the line being edited.

Version A1.V02 - March 1985

The following control characters are used for editing effects:

↑A	Cursor to beginning of line
↑B	Cursor left (back)
↑D	Delete character under cursor
↑E	Cursor to end of line
↑F	Cursor right (forward)
↑K	Delete to end of line
↑R	Retype line
↑U	Delete entire line
DEL	Delete character to left of cursor
RET	Flush buffer

Revised Installation Procedures

The PS 300 distribution tape now contains NETEDIT executables as well as source files. This simplifies the installation procedure for sites where no modifications to the source or data files are planned, or where no Pascal compiler is available. Note that the executables were built on a VAX 780 running VMS 3.7, and may not work properly on other versions of the hardware or software.

The procedure for installing NETEDIT without rebuilding it entirely is as follows:

1. Set default to Netedit subdirectory in the A2.V01 subdirectory.

2. Edit NETUSER.COM and change the definition of NETROOT (marked !*INSTALL-DEPENDENT) to the name of the directory created. Make sure this file is readable and executable by all users. See comments in Netuser.Com "Site Customization of Netuser.Com."

3. Copy the empty user log file, NETEDIT0.USR to NETEDIT.USR. Set the protection on this file so that it is writable by all users.

The procedure to install the editor by rebuilding the executables is essentially unchanged. Note that there is an additional !*INSTALL-DEPENDENT parameter in NETBUILD.COM which specifies the directory where the PS 300 Pascal GSR library resides. If you do not have this library, you may use the dummy library supplied on the tape. See comment in NetBuild.Com "Site Customization of NetBuild.Com."

Source files for the user-written functions used by NETEDIT, along with a command file to build the .300 files which may be downloaded to the PS 300, have been provided. However, to rebuild the user-written functions, you must have the Motorola 68000 cross software, which is not supplied by Evans & Sutherland.

KNOWN "BUGS" IN THE PS 300 DOCUMENT SET

Volume	Section	Page	Change Description
2A	Hands-on Experience	2	The command to set the line-drawing speed for CSM displays should read: SEND TRUE TO <1> CSM;
2A	Command Language	11	The command to translate the tire primitive to the front left location should read: Tran_FL_Tire := TRANSLATE BY .5415,-.1598, .3357 APPLIED TO Rot_FL_Tire;
		14	The TRANSLATE command in the definition of the front left snow tire (FL-tire) should read: TRANSLATE BY .5415, -1598, .3357; The remainder of the structure is correct.
		20&23	Figures 4 and 5 are reversed.
		23-28	In BEGIN_STRUCTURE ... END_STRUCTURE commands, there should be no semicolon following "BEGIN_STRUCTURE".
2A	Function Networks 1	4	A new page is supplied to call out the names of interactive nodes in the figure.
		32	The integer 2 should not go to <1>Roty, but to <1>Timer. The command should read: SEND FIX(2) TO <1>Timer;
2A	Viewing Operations	43	In the definition of Top, the viewport command should create a viewport with dimensions 0 to 1 horizontally and vertically. It now creates a viewport from 0 to -1 horizontally and vertically. The viewport command in Top should read: VIEWPORT HORIZONTAL=0:1 VERTICAL=0:1;
		45	The names in the last two commands on this page are wrong, they should be Nonsquare_Window and NonSquare. The last two commands should read: DISPLAY Nonsquare_Window; REMOVE NonSquare;

KNOWN "BUGS" IN THE PS 300 DOCUMENT SET

Volume	Section	Page	Change Description
2B	Conditional Referencing	13	IF LEVEL_OF_DETAIL nodes accept integers from 0 to 32767, not from 0 to 14, as stated near the bottom of this page.
		17	Select LEVEL OF DETAIL from the Tutorial Demonstration Programs, not ANIMATED CYLINDER.
2B	Function Networks II	3	A new page is supplied to call out the names of interactive nodes in the figure.
2B	Function Networks II	22	Last command should read: CONNECT Switch7<3> : <1>Rot_Lt_Elbow;
		39&40	Every command must be followed by a semicolon.

2B	Text Modeling	20	Near the top, the command to display the scaled limerick now reads:
			DISPLAY Limerick;
			It should read:
			DISPLAY Scale_Block;

2B	Rendering Operations	17	A replacement page is supplied to correct coordinate values.
		20	In "Object" on the bottom of page 20, polygon 2 is an {outer}, and polygon {5} should be defined as follows: {non-coplanar} POLYGON 1,1,0 1,1,1 1,-1,1 1,-1,0;
3A	Command Summary	Bspline	The node diagram shows inputs <1>, <2>, and <3>. It should show input <1> and input <i>. A real number sent to input <i> changes the knots in the curve. A 2D, 3D, or 4D vector sent to input <i> changes the vertices in the curve.
		Rational Bspline	The node diagram shows inputs <1>, <2>, and <3>. It should show input <1> and input <i>. A real number sent to input <i> changes the knots in the curve. A 2D, 3D, or 4D vector sent to input <i> changes the vertices in the curve.
		Labels	A 3D vector can be sent to input <i> to change the starting location of the i-th label. This is not currently shown.

KNOWN "BUGS" IN THE PS 300 DOCUMENT SET

Volume	Section	Page	Change Description
3A	Command Summary	Polynomial	The node diagram shows inputs <1> and <2>. It should show inputs <1> and <i>. An integer sent to input <1> changes the number of chords in the curve and a vector sent to input <i> changes the curve coefficients. The present documentation has this information reversed.
		Rational Polynomial	The node diagram shows inputs <1> and <2>. It should show inputs <1> and <i>. An integer sent to input <1> changes the number of chords in the curve and a vector sent to input <i> changes the curve coefficients. The present documentation has this information reversed.

		XFORM	The 4x4 matrix shown in the example does not contain enough elements and is improperly formatted. In addition, the semicolon is missing from the END_S command. The commands should read:
			
			`TRAN := BEGIN_S {To be used while getting transformed data}`
			`MATRIX_4x4 1,0,0,0  0,1,0,0  0,0,1,0  0,0,0,1;`
			`INSTANCE OF OBJ;`
			`END_S;`

3A	Function Summary	F:ACCUMULATE	The default values on inputs <3> and <4> are not given. Input <3> defaults to 0 and input <4> defaults to 1.
		F:LINEEDITOR	Output <4> is incorrectly documented as an input to <append> of a Characters node. It should read input <delete> of a Characters node.
		F: MCONCATENATE(n)	This function is misnamed. Its correct name is F:MCAT STRING(n).
		F:STRING_TO_NUM	This function converts a string of digits to a real number. Missing from the current documentation is output <2>. This is a Boolean value which is TRUE if the string received can be converted and false otherwise.
		FKEYS	This initial function instance is listed as an Output function. In fact, it is an Input function.
		TSCSM	This has been erroneously documented as TECSM.

KNOWN "BUGS" IN THE PS 300 DOCUMENT SET

Volume	Section	Page	Change Description
3A	Function Summary	F:SEND	Previously undocumented. A new page is supplied.
		F:LIST	Previously undocumented. A new page is supplied.
		ONBUTTONLIGHTS	Previously undocumented. A new page is supplied.
		CSM	Previously undocumented. A new page is supplied.
5	Firmware and Host Software	4	Under "Creating and Downloading the SITE.DAT," Step 3, the correct demuxing character and routing byte to begin the SITE.DAT file is ↑\: (↑ represents the CONTROL value of the \ key). The user-defined commands are entered after these characters and the file must end with the demuxing character and routing byte: ↑\;.
5	Local Data Flow	5-38 (5-62 IMB hosts only)	Table 5-1 Routing Byte Definitions is incomplete and may be misleading. A new page is supplied.

Enhancements in Graphics Firmware Version A2.V01

- This release of the graphics firmware provides the new Writeback feature. The Writeback Feature allows displayed transformed data to be sent back to the host. This feature provides a Writeback command and a Writeback function.

  The Writeback command creates a WRITEBACK operation node and enables the data structure below the node for writeback operations. When the Writeback node is activated, writeback is performed for name1 (the name of the structure for which writeback is applied). A default WRITEBACK operation node is created by the system at initialization time.

  The Writeback Function is initialized by the system and is used to send encoded writeback data to user function networks. This function is not activated by the normal input queue triggering mechanism. It is activated by sending a TRUE to any writeback operation node in a display structure.

  Writeback is described completely in the Writeback Feature User's Guide, included with this release.

- PVecMax (PVCMax-FORTRAN) has been added to the GSRs. This procedure sets the maximum component of a block-normalized vector list, so that multiple calls may now be made to PVecList for block-normalized vectors.

Modifications in the Graphics Firmware

- Changes to BUTTONSIN (PS 350 Only)

  The initial function instance BUTTONSIN has two new inputs.

  Integer <2> Enable/Disable Bit Mask
  Default FIX(-1) all buttons enabled.

  Boolean <3> TRUE - enable use of bit mask
  FALSE - disable use of bit mask.
  Default FALSE

  The Buttonsin bit mask is a mapping of the bits of a 32-bit integer to the individual buttons. The Most Significant Bit (sign bit) maps to button #1; the least significant bit maps to button #32.

Most Significant Bit ↓ (bit 31) — Least Significant Bit ↓ (bit 0)

Bits of the Integer	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Button Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32

If the bit is set (=1), the button is enable. If the bit is off (=0), the button is disabled.

- Changes to ONBUTTONLIGHTS and OFFBUTTONLIGHTS (PS 350 Only)

The initial function instance ONBUTTONLIGHTS/OFFBUTTONLIGHTS has one new input.

New input
<2> Boolean
TRUE - interpret integer on input <1> as a bit mask.
FALSE - interpret integer on input <1> as a button number.

The ONBUTTONLIGHTS/OFFBUTTONLIGHTS bit mask is a mapping of the bits of a 32 bit integer to the individual buttons. The most significant bit (sign bit) maps to button #1; the least significant bit maps to button #32. If the bit is set (=1) the button light is on.

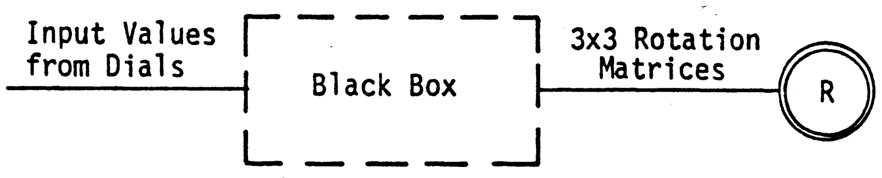
CHANGE PAGES TO THE DOCUMENT SET OTHER THAN THE COMMAND SUMMARY, THE FUNCTION SUMMARY, AND GRAPHICS SUPPORT ROUTINES

CONVERTING INPUT DEVICE VALUES TO UPDATE AN INTERACTION NODE

The first step to selecting the appropriate function to convert input values into values that can update an interaction node is to identify the type of values needed by the node. To understand this, look at the the most common graphics transformations--rotation, scaling, and translation.

Rotations and scales are done with 3x3 matrices; translations are specified with a 2- or 3-dimensional vector. It makes sense, then, that the type of data used by a rotation or scale node is a 3x3 matrix, and the data type for a translation node is a vector.

Your task, if you are trying to rotate part of a model, is to find a way to make an input device, such as a dial, send the correct 3x3 matrices to a rotate node. In this module, this process will be represented by a "black box" (Figure 2) that takes one kind of value and changes it into another kind.



1AS0527

Figure 2. The "Black Box"

In the "Hands-On Experience" module, you created Diamond by specifying a 45 degree rotation of Square. You did not need to work out what the 3x3 matrix for 45 degrees was. Whenever you use a command to create a rotate or scale node (such as Diamond), you only have to specify an angle using a real number value and the PS 300 automatically creates the associated 3x3 matrix.

Once the node is created, however, you can only update it with the type of data it accepts--in this case, a 3x3 matrix. For example, look at the robot display tree again (Figure 3) the names for the interactive nodes are supplied so you can refer to them.

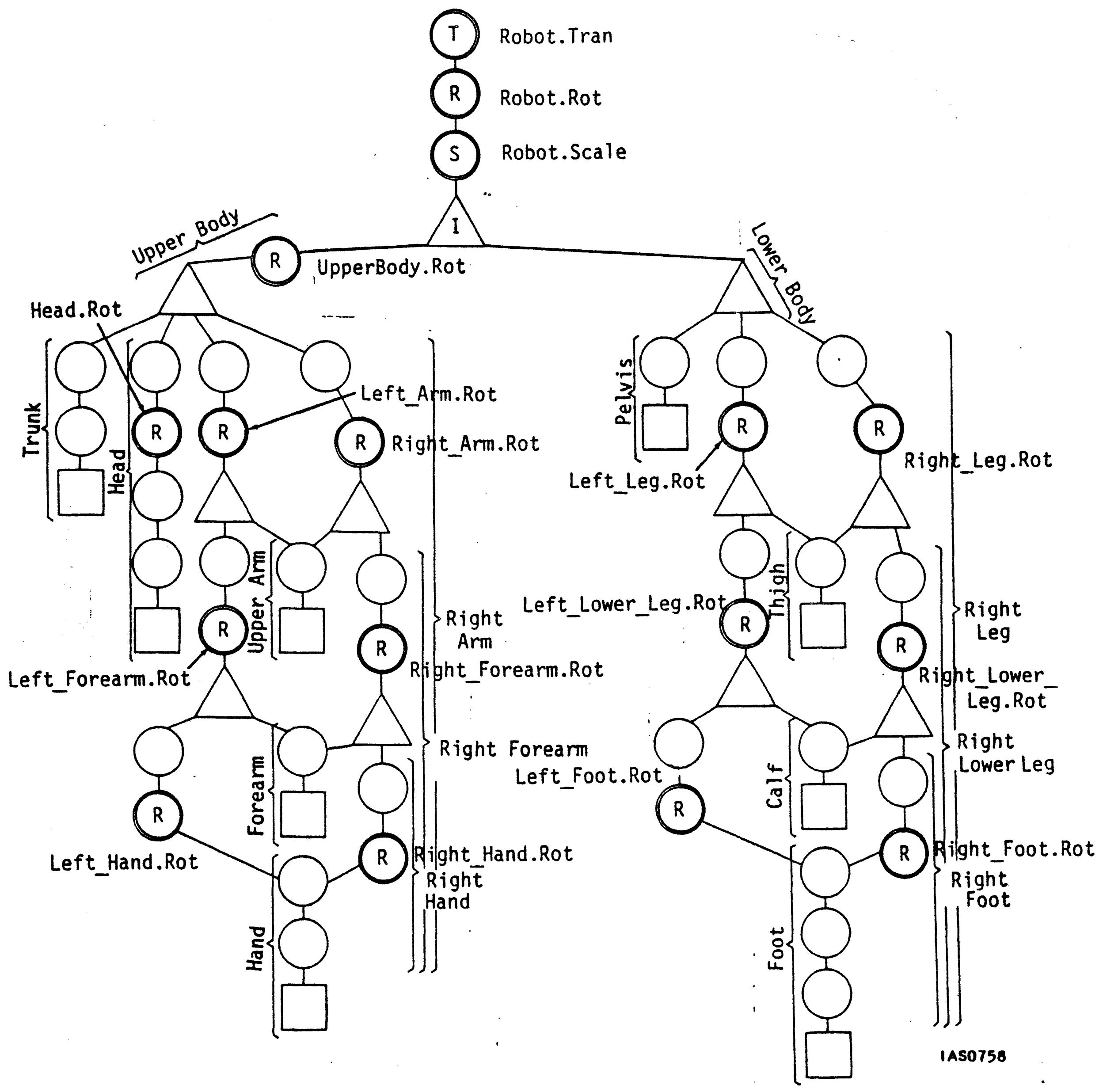


Figure 3. Interactive Nodes in Robot Display Tree

MAKING A SINGLE INPUT DEVICE CONTROL MULTIPLE INTERACTIONS

In "Function Networks I," you constructed a function network for the display tree shown in Figure 1.

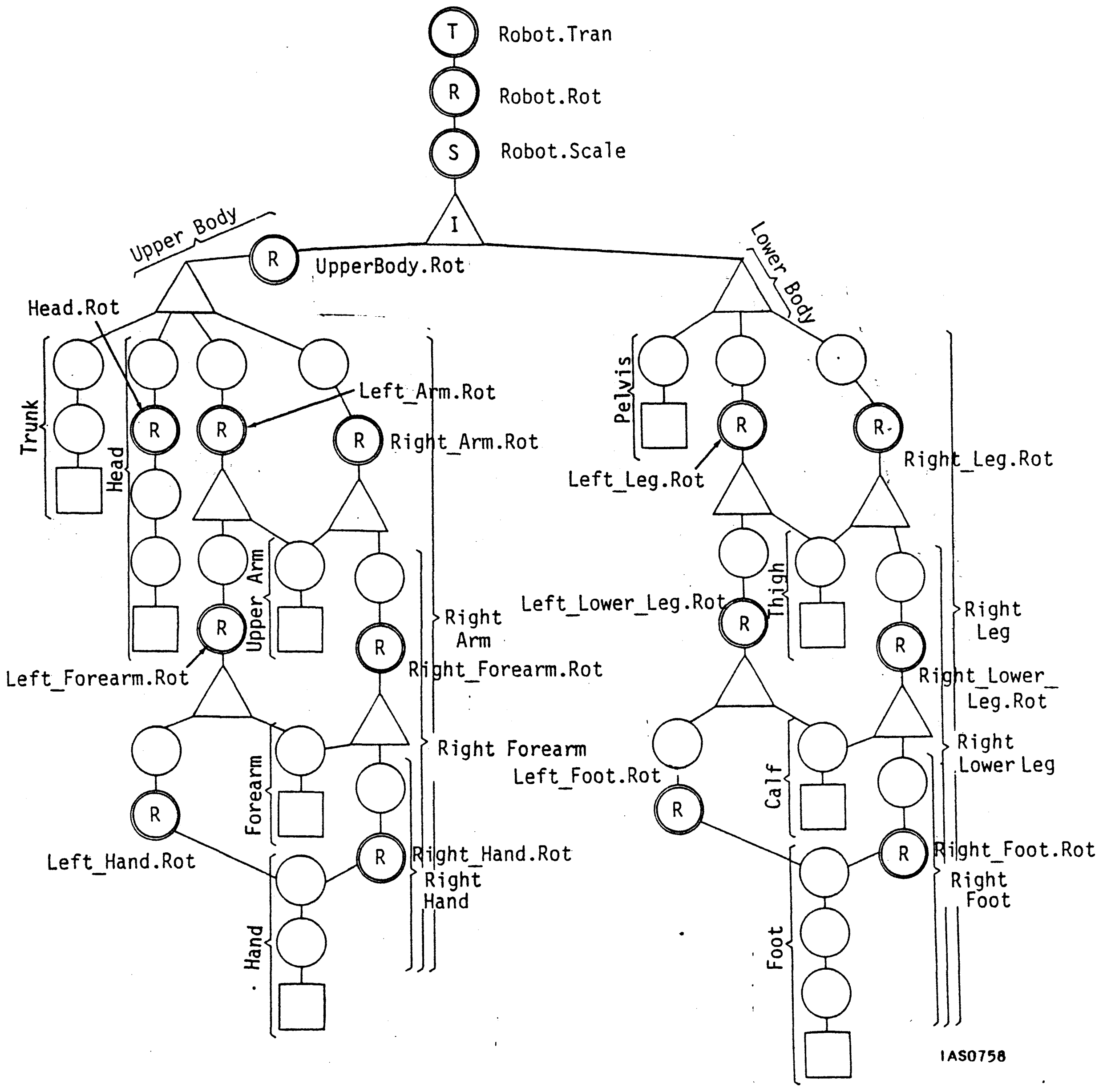


Figure 1. Robot Display Tree

This function network supplied interactions for the top three nodes of the display tree: Robot.Scale, Robot.Rot, and Robot.Tran. Seven dials were required to manipulate the robot: three to rotate it in the X, Y, and Z planes, three to translate it in X, Y, and Z, and one dial to scale the model.

Only one free dial remains, but no other interactive nodes in the robot display tree have yet been connected to functions. To supply X, Y, and/or Z rotations for all the other interactive nodes would require dozens of other dials. This section illustrates how to solve this problem by making one set of eight dials perform like many sets.

The first step in doing this is to determine exactly how many additional dials you will need by deciding how many more interactions in the model you want to control. In addition to Robot.Rot, the robot has 14 rotation nodes. Ten of them require three dials each (three rotations for X, Y, and Z). The two nodes for elbows and the two for knees only use X rotations, requiring only one dial each. The result is a total of 34 additional interactions. To handle these interactions, each dial would have to be connected to about six nodes.

There is nothing to prevent you from connecting a dial to more than one destination. For example, you could hook dial 1, already updating X rotations for the Robot.Rot node, to other rotate nodes. But of course turning that one dial would cause multiple unrelated updates.

Following is one way the dials might logically be assigned to control the interactions.

In Mode 1, the dials would work as presently assigned:

Whole model:	1. Xrot	2. Yrot	3. Zrot	4. Scale
	5. Xtran	6. Ytran	7. Ztran	8. Not Assigned

Mode 2:

Head:	1. Xrot	2. Yrot	3. Zrot	4. Not Assigned
Trunk:	5. Xrot	6. Yrot	7. Zrot	8. Not Assigned

Mode 3:

Right arm:	1. Xrot	2. Yrot	3. Zrot	4. Elbow Xrot
Left arm:	5. Xrot	6. Yrot	7. Zrot	8. Elbow Xrot

Given the following object (Cube):

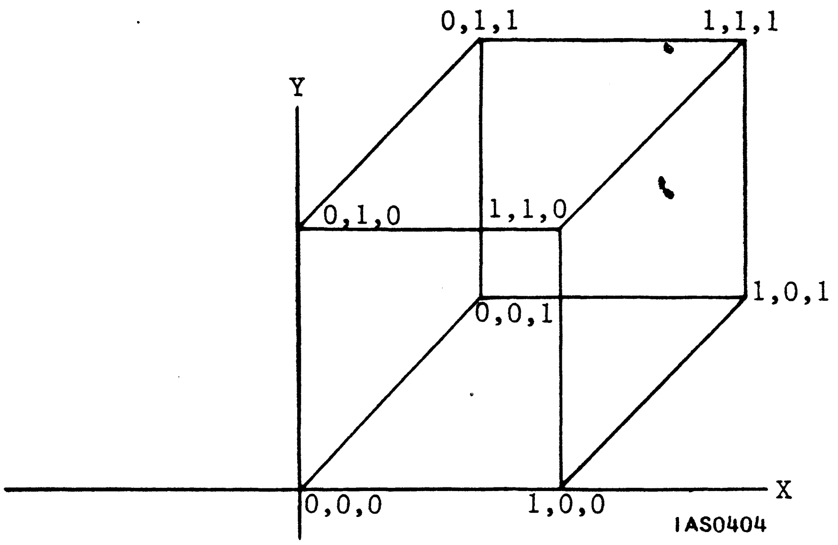


Figure 12. Cube

A correct syntax to define this object is as follows:

```
Cube :=    POLYGON 0,0,0  0,1,0  1,1,0  1,0,0
           POLYGON 1,0,0  1,1,0  1,1,1  1,0,1
           POLYGON 1,1,1  0,1,1  0,0,1  1,0,1
           POLYGON 0,1,1  0,1,0  0,0,0  0,0,1
           POLYGON 0,1,1  1,1,1  1,1,0  0,1,0
           POLYGON 1,0,0  1,0,1  0,0,1  0,0,0;
```

Associating Outer and Inner Contours With COPLANAR

A polygon that represents a face of an object is called an **outer contour**. Some polygons, known as **inner contours** represent cavities, holes, or protrusion sites in an object.

For the PS 340 to interpret inner contours properly, two things must be done. One is to observe the vertex-ordering convention for inner and outer contours. The other is to use the COPLANAR option in the POLYGON clause to associate inner and outer contours.

The vertex ordering rule for inner and outer contours is as follows: vertices of inner contours must run in the opposite sense to the corresponding outer contour. For a solid this implies that the vertices of an inner contour run counterclockwise while outer contours run clockwise when viewed.

The vertices of the following triangular polygon face (outer contour) with a hole in it (inner contour) are ordered as follows.

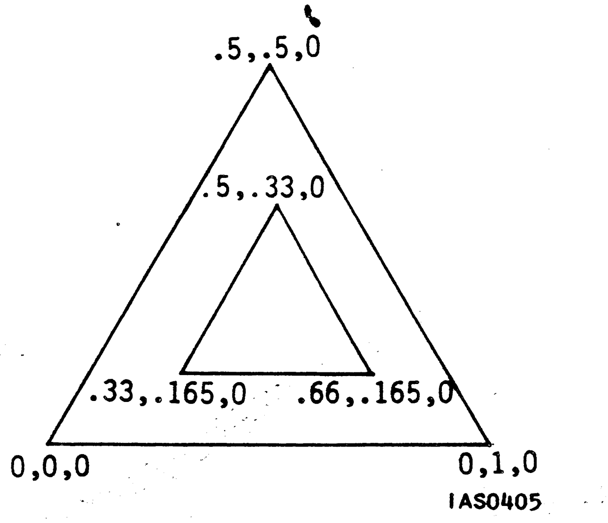


Figure 13. Surface With Inner/Outer Contours

A POLYGON command syntax for this object is :

```
Object :=  POLYGON 0,0,0  .5,.5,0  1,0,0 {outer contour}
           POLYGON COPLANAR .5,.33,0  .33,.165,0  .66,.165,0;
           {inner contour}
```

Note that the vertices for the inner contour in the above example are listed in the opposite order of those of the outer contour.

5.2 DATA RECEPTION AND ROUTING NETWORK

F:CIROUTE

Once data have passed through either instance of F:DEPACKET (described in the previous chapter), the next function to receive it is F:CIROUTE. F:CIROUTE has two instances, one for count mode and one for escape mode. They are functionally very similar, and only the count mode instance, CIROUTE0 will be described. CIROUTE0 examines the first character it receives (the character following the count bytes in count mode and the character following the <FS> character in escape mode) to determine where the packet message is to be sent. These characters are "routing" bytes, and are used to select the appropriate channel for data in the PS 300. Data channels include lines to the terminal emulator, the PS 300 command interpreter, the Disk writing function, the Raster function (for PS 340 systems), and other system functions. A base character (defined on Input <2> of CIROUTE0) is subtracted from this routing character before it is used to select the output channel. The base character defaults to the character zero ("0").

```
                              _____________________
                             |                     |
Qpacket,Qmorepacket-------->| <1>             <1> |------> Qinteger
         Qreset             |                     |
                             |                 <2> |------> Qpacket,
                             |                  .  |        Qmorepacket
           Qstring -------->| <2>C '0' Base     .  |
                             |                  .  |
           Qprompt -------->| <3>C            <n> |------> Qpacket,
           Qreset            |                     |        Qmorepacket
                             |                     |
           Qinteger ------->| <4>C                |
                             |       CIROUTE0      |
                             |       F:CIROUTE     |
                             |     (count mode)    |
                             |_____________________|
```

The definitions for the inputs and outputs for F:CIROUTE are described in Chapter 6 of this guide.

5.2.1 Routing Byte Definitions

The following table defines the routing bytes and channel parameters for assessing internal PS 300 communication channels.

Table 5-1. Routing Byte Definitions

CIROUTE Output	Routing Byte	Channel Parameter	Description
3	0	1	Parser/Command Interpreter
4	1	2	Command Interpreter via READSTREAM
5	2	3	6-bit binary
6	3	4	Reset network for GSRs
7	4	5	Reserved
8	5	6	Reserved
9	6	7	Download channel for user-written functions
	•		
	•		
	•		
13	:	11	Write ASCII data to diskette
14	;	12	Close file
15	<	13	Write binary data to diskette
16	=	14	Reserved
17	>	15	Channel to terminal emulator
18	?	16	Host message control
19	@	17	Who (used by PSETUP)
20	A	18	Reserved
21	B	19	Raster

NOTE

('?') is the HOST_MESSAGE request channel. <SOP>? followed by ASCII (1 or 2) requests a single message or multiple messages from HOST MESSAGEB.

('@') any message sent on this route triggers the WHO function. (Refer to the PS 300 Host-Resident I/O Subroutine Manual for information on the WHO function.)

5.1.1 Data Flow Overview

The PS 300 accepts data from the IBM host line through the General Purpose Interface Option (GPIO) card. These data will be in two forms; either data for the terminal emulator, or graphical data that have been sent from the host using the cross-compatibility software. The GPIO differentiates between data designated for the terminal emulator and graphics data packaged in the Write Structured Field (WSF) envelopes. The GPIO puts TE data directly into a Screen Buffer in Mass Memory. Graphics data are intercepted by the GPIO for unpacking and repackaging into Qpackets. Routing information is always included at the head of any WSF command. The routing information and the first 238 bytes of data are put into a Qpacket by the GPIO. All subsequent data within the same WSF command are placed into Qmorepackets. When a WSF command is filled to capacity, or a routing change is required, the present WSF command is terminated and a new WSF command is started by a PS 300 low-level communication routine.

The packets of graphical data are passed to the data reception function, F:F_I1_IBM (F:F_12_IBM). IBMI1$ (an instance of F:F_I1_IBM) allocates new mass memory packet buffers and puts them on a linked list for subsequent use by the GPIO. IBMI1$ passes data through to F:CIROUTE.

F:CIROUTE

CIROUTE0 examines the first character it receives (the character following the count bytes in count mode) to determine where the packet message is to be sent. These characters are "routing" bytes, and are used to select the appropriate channel for data in the PS 300. (Data channels can be chosen by the use of the parameter in the PMuxG GSR Utility Routine. Standard GSR and PSIO calls include embedded routing bytes.) Data channels include lines to the terminal emulator, the PS 300 command interpreter, the disk writing function, the raster function (for PS 340 systems), and other system functions. A base character (defined on Input <2> of CIROUTE0) is subtracted from this routing character before it is used to select the output channel. The base character defaults to the character zero ("0"). The definitions for the inputs and outputs for F:CIROUTE are described in Chapter 6 of this guide.

```
                             ______________________________
Qpacket,Qmorepacket------->|<1>                       <1>|------> Qinteger
         Qreset            |                              |
                           |                          <2>|------> Qpacket,
                           |                           .  |        Qmorepacket
           Qstring ------->|<2>C '0' Base              .  |
                           |                           .  |
           Qprompt ------->|<3>C                      <n>|------> Qpacket,
           Qreset          |                              |        Qmorepacket
           Qinteger ------>|<4>C                          |
                           |      CIROUTE0                |
                           |      F:CIROUTE               |
                           |______________________________|
```

5.1.2 Routing Byte Definitions

The following table defines the routing bytes and channel parameters for assessing internal PS 300 communication channels.

Table 5-1. Routing Byte Definitions

CIROUTE Output	Routing Byte	Channel Parameter	Description
3	0	1	Parser/Command Interpreter
4	1	2	Command Interpreter via READSTREAM
5	2	3	6-bit binary
6	3	4	Reset network for GSRs
7	4	5	Reserved
8	5	6	Reserved
9	6	7	Download channel for user-written functions
	•		
	•		
	•		
13	:	11	Write ASCII data to diskette
14	;	12	Close file
15	<	13	Write binary data to diskette
16	=	14	Reserved
17	>	15	Channel to terminal emulator
18	?	16	Host message control
19	@	17	Who (used by PSETUP)
20	A	18	Reserved
21	B	19	Raster

NOTE

('?') is the HOST_MESSAGE request channel. '?' followed by ASCII (1 or 2) requests a single message or multiple messages from HOST MESSAGEB.

('@') any message sent on this route triggers the WHO function.

F:IBM_KEYBOARD

```
                 ________________________
                |                        |
                |     F:IBM_KEYBOARD     |
                |                        |
Qpacket----->   |<1>                  <1>|----->Qpacket
                |                        |
QBoolean --->   |<2>                  <2>|----->Qinteger
                |                        |
                |                     <4>|----->Qpacket
                |                        |
                |                     <3>|----->Qpacket
                |                        |
                |                     <5>|----->Qpacket
                |                        |
                |                     <6>|----->QBoolean
                |                        |
                |                     <7>|----->Qpacket
                |                        |
                |                     <8>|----->QBoolean
                |                        |
                |                    <10>|----->QBoolean
                |     (IBMKBD1)          |
                |                    <11>|----->QBoolean
                |________________________|
```

F:IBM_KEYBOARD accepts character packets from the keyboard on input <1> and, based on the mode selected by the mode keys (either the LINE LOCAL key or the HOST, LOCAL and COMMAND keys, depending on the type of keyboard used), outputs packets for use by the function network, the line editor, or an IBM host. Packets of characters for the function KEYBOARD are output on output <1>. Qintegers to be sent to the function FKEYS are output on output <2>. Qpackets of characters to be sent to the function SPECKEYS are output on output <3>. Qpackets of characters for the line editor are output on output <4>. Qpackets of IBM scan codes for an IBM host are output on output <5>. A QBOOLEAN TRUE used to trigger the hardcopy functions is output on either output <6>, output <10>, or output <11>, based on the mode of the keyboard.

A TRUE used to trigger the loading of the IBM 3250 function network is output on output <7> when IBM 3250 mode is selected while in SETUP mode.

A TRUE used to trigger the deletion of the IBM 3250 function network is output on output <8> when the PS 300 mode is selected while in SETUP.

Input <2> accepts a Boolean that indicates which type of keyboard is being used.

True = IBM style keyboard
False = VT100 style keyboard

F:IBM_SETUP

```
                 _______________________
                |                       |
QBoolean  --->  |<1>   F:IBM_SETUP      |
                |      (IBMSETUP1)      |
                |                       |
Qinteger  --->  |<2>                    |
                |                       |
Qinteger  --->  |<3>                    |
                |                       |
                |_______________________|
```

F:IBM_SETUP is used to change the parameters used by the IBM communications. Input <1> accepts an integer that specifies the maximum number of packets that can be in the pool of empty input packets.

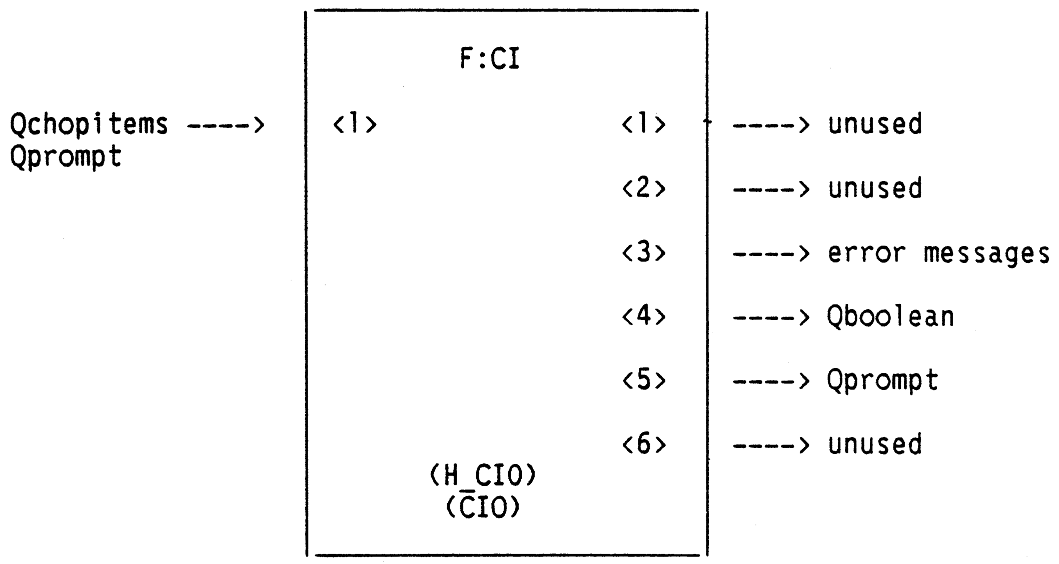
F:IBM_SETUP is used to change certain values used by the IBM communications.

Input <1> is used to trigger the function.

Input <2> is used to specify the number of empty I/O input packets that are to be maintained in the I/O input pool.

Input <3> is used to specify the device address when an IBM 3250 interface is being used.

F:CI



This function interprets commands, creating display structures and function networks. It receives input either from a chop/parse function or a Readstream function (if using the GSRs).

A single parameter is given when this function is instanced (for example H CI0:=F:CI(4);). This parameter is the "CINUM" and is used to identify all names and connections this CI makes. When the CI receives an INIT command, it destroys only those connections it has made and only those structures associated with the names which have its CINUM.

Note: A name is created when that name is referenced for the first time, even if it has no associated structure. The CI that created the name is the "owner" of that name, even if the entity it refers to is created by another CI.

Note: Each function has an output <0> that is used to send error messages (such as illegal input error messages). The connection from this output is made automatically by the CI that creates the function. The CI finds the appropriate error function to connect output <0> to by looking on its own output <3>.

Output 4 sends out a Qboolean with a TRUE value when an INIT command is entered. This output is connected to the initial function CLEAR_LABELS to clear out the labels on the keyboard and dials.

7.5.3 Command Status Command

The command:

COMMAND STATUS;

directs the command interpreter to print the status of the command stream. The message output lists the number of open BEGIN...END and BEGIN_STRUCTURE...END_STRUCTURE commands, and indicates if the privileged state is operative. The message also indicates if the optimize structure model is in effect.

7.5.4 Reboot Command

The command

```
   REBOOT password;
```

reboots the PS 300 as if from power-up. If no password has been setup, then any character string will do. Otherwise entering an incorrect password will give an error message. The REBOOT command can appear anywhere; it can occur within BEGIN...END and BEGIN_STRUCTURE...END_STRUCTURE as well as without. It may be named or not. However, it cannot be within a quote or comment.

The command causes the PS 300 to reboot just as if it had been powered up (starts the confidence tests at "A", etc.).

7.5.5 Set Priority

The command

```
   Set Priority of name to i;
```

sets the execution priority of a function (name) to some integer (i) between 0 and 15. All user instancible functions and most functions instanced by the system at boot time have a default value of 8. Lowering a function's priority number raises its priority and causes it to run before any functions with a larger number. A typical use of this command is to give to a function a priority number greater than 8 so it runs only when no other functions are running (i.e. functions at default priority 8). Assigning priority numbers less than 8 could be potentially very "dangerous," since their execution could lock up the system.

Since this command will affect the execution of other functions in a function network, careful consideration must be given to its use. E&S does not recommend the use of this procedure by anyone who does not have a complete understanding of functions and their interrelationships.

7.5.6 Notes On Using the CONFIGURE Mode

E&S reserves the right to change the content of the CONFIG.DAT file and the implementation of the CONFIG.DAT file without prior notice. Use of any named entities or networks instanced in CONFIGURE mode that have names identical to any names found in the CONFIG.DAT file will result in unpredictable system behavior. E&S will not use any names that are preceded with the three characters CM_.

TABLE 9-1 PS 300 TRAPS and Their Meanings

NUMBER	DEFINITION
0	Not enough available memory to come up or handle request.
1	E&S firmware error.
2	Memory corrupted or over-written (could be caused by UWF).
5	Attempt to wait on queue when function is waiting on another device (CLOCK, I/O)(could be caused by UWF).
6	System errors (see Table 3).
8	Mass memory error if address on LEDs is between 200 and 300; unexpected interrupt on a vector with no routine, if address is between 300 and 400. For example, if address on LEDs is 22C, error occurred on memory card 200000-300000. If address is 23C, error occurred on memory card 300000-400000 and so forth.
9	Utility routine was called which was not included in system link.
10	Memory corrupted or over-written (could be caused by UWF).
11	E&S firmware error.
12	Pascal in-line runtime error: usually caused by Case statement in Pascal with no Otherwise clause (could be caused by UWF).

Version A2.V02

F:READDISK

```
                    ___________________________
                   |                           |
                   |         F:READDISK        |
                   |                           |
    Qpacket  ----> |<1>                     <1>| ---->  Qpacket
                   |                           |
    QBoolean ----> |<2>                     <2>| ---->  QBoolean
                   |                           |
                   |(Readascii1 for ASCII file)|
                   |(Readbinary1 for binary)   |
                   |___________________________|
```

This function reads a file from the floppy disk and sends the data out output <1> in Qpackets. Input <1> accepts a Qpacket of 1 to 8 characters specifying the name of the file to be read. All disk drives are searched for the file until found; if the file is not found, an error message is produced.

A True on input <2> tells the function to delete the file after reading. Input <2> is a constant input queue and is initialized to False.

A True is output from <2> when the file is found and read successfully. A False is output when the file is not found.

Note: The file name sent on input <1> should not include the file extension. The file on the disk must have the extension ".DAT".

CHANGE PAGES TO THE COMMAND SUMMARY, THE FUNCTION SUMMARY, AND GRAPHICS SUPPORT ROUTINES

(Change pages exclusive to the Rendering Option are supplied with the PS 390 Rendering Release Notes.)

FORMAT

```
name := LOAD VIEWport HORizontal = hmin:hmax
               VERTical = vmin:vmax
               [INTENsity = imin:imax] [APPLied to name1];
```

DESCRIPTION

The LOAD VIEWPORT command for the PS 350 loads a viewport and overrides the concatenation of the previous viewport. As with the standand PS 300 VIEWPORT command, it specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines. It affects all objects below the node created by the command in the display tree.

PARAMETERS

hmin,hmax,vmin,vmax – The x and y boundaries of the new viewport. Values must be within the −1 to 1 range.

imin,imax – Specifies the minimum and maximum intensities for the viewport. **imin** is the intensity of lines at the back clipping plane; **imax** at the front clipping plane. Values must be within the 0 to 1.

name1 – The name of the structure to which the viewport is applied.

DEFAULT

The initial viewport is the full PS 300 screen with full intensity range (0 to 1) using the standard PS 300 Viewport command.

```
VIEWport HORizontal = -1,1 VERTical = -1,1 INTENsity = 0:1;
```

(continued)

NOTES

A new VIEWport is not defined relative to the current viewport, but to the full PS 300 screen.

If the viewport aspect ratio (vertical/horizontal) is different from the window aspect ratio (y/x) or field-of-view aspect ratio (always 1) being displayed in that viewport, the data displayed there will appear distorted.

DISPLAY TREE NODE CREATED

This command creates a load viewport operation node that has the same inputs as the standard viewport operation node. The matrix contained in this node is not concatenated with the previous viewport matrix.

NOTES ON INPUTS

1. For 2x2 matrix input, row 1 contains the **hmin,hmax** values and row 2 the **vmin,vmax** values.

2. For 3x3 matrix input, column 3 is ignored (there is no 3x2 matrix data type), rows 1 and 2 are as for the 2x2 matrix above, and row 3 contains the **imin,imax** values.

Version A2.V01

FORMAT

```
name := RAWBLOCK i;
```

DESCRIPTION

Used to allocate memory that can be directly managed by a user-written function or by the physical I/O capabilities of the Parallel or Ethernet Interfaces.

PARAMETERS

i - bytes available for use.

NOTES

1. The command carves a contiguous block of memory such that there are "i" bytes available for use.
2. The block looks like an operation node to the ACP. The descendant alpha points to the next long word in the block. What the ACP expects in this word is the .datum pointer of the alpha block. (The datum pointer points to the first structure to be traversed by the ACP. This is the address in memory where the data associated with a named entity is located.)
3. To use this block, the interface or user-written function fills in the appropriate structure following the .datum pointer. When this is complete, it changes the .datum pointer to the proper value and points to the beginning of the data. After the ACP examines this structure, it displays the newly-defined data. (Use the ACPPROOF procedure to change the .datum pointer with a user-written function.)
4. More than one data structure at a time can exist in a RAWBLOCK. It is up to the user to manage all data and pointers in RAWBLOCK.
5. A RAWBLOCK may be displayed or deleted like any other named data structure in the PS 300. When a RAWBLOCK is returned to the free storage pool, the PS 300 firmware recognizes that it is a RAWBLOCK and does not delete any of the data structures linked to RAWBLOCK.

DISPLAY TREE NODE CREATED

Rawblock data node.

FORMAT

```
name := SET BLINKing switch
            [APPLied to name1];
```

DESCRIPTION

This command turns blinking on and off. It affects all objects below the node created by the command in the display tree.

PARAMETERS

switch – Boolean value. TRUE indicates that blinking will occur in the displayed objects. FALSE turns blinking off.

name1 – The name of the structure that will be affected by the command.

NOTE

PS 330 style blinking, done via the SET RATE and IF PHASE ON/OFF commands, where blinking is tied to the update rate rather than the refresh rate, will still work, but since the update rate in the PS 350 may be slower, the visual result may be different.

DISPLAY TREE NODE CREATED

This command creates a set blinking on/off operation node in the display structure that determines whether blinking will occur in the objects positioned below it in the display structure.

INPUTS FOR UPDATING NODE

The blinking on/off operation node can be modified by sending a Boolean value to input <1>.

FORMAT

```
name := SET BLINK RATE n
            [APPLied to name1];
```

DESCRIPTION

This command specifies the blinking rate in refresh cycles to be applied to all objects below the node created by the command in the display tree.

PARAMETERS

n – An integer designating the duration of the blink in refresh cycles. The blinking data will be on for n refreshes and off for n refreshes.

name1 – The name of the structure to which the blinking rate is applied.

NOTE

PS 330 style blinking, done via the SET RATE and IF PHASE ON/OFF commands, where blinking is tied to the update rate rather than the refresh rate, will still work, but since the update rate in the PS 350 may be slower, the visual result may be different.

DISPLAY TREE NODE CREATED

This command creates a set blinking rate operation node in the display tree that specifies the blinking rate for all objects below it.

INPUTS FOR UPDATING NODE

The node can be modified by sending an integer to input <1> which will change the blinking rate.

FORMAT

```
name := SET LINe_texture [AROUnd_corners] pattern
                            [APPLied to name1];
```

DESCRIPTION

Specifies the line texture pattern to be used in drawing the vector lists that appear below the node created by this command. There are up to 127 hardware-generated line textures possible. The parameter **pattern** is an integer between 1 and 127. The desired line texture is indicated by the setting or clearing of the lower 7 bit positions in **pattern** when represented in binary. An individual pattern unit is 1.1 centimeters in length. Some of the more common patterns and their corresponding bit settings are shown below:

Pattern	Bit representation	Line Texture repeated twice	
127	1111111	--------------	Solid
124	1111100	----- -----	Long Dashed
122	1111010	---- - ---- -	Long Short Dashed
106	1101010	-- - - -- - -	Long Short Short Dashed

PARAMETERS

AROUnd_corners – Boolean value used to set a flag to indicate if the specified line texture should continue from one vector to the next. If AROUnd_corners is TRUE, the line texture will continue from one vector to the next through the endpoint. If AROUnd_corners is FALSE, the line texture will start and stop at vector endpoints.

pattern – An integer between 1 and 127 that specifies the desired line texture. When **pattern** is less that 1 or greater than 127, solid lines are produced.

name1 – The name of the structure to which the line texture is applied.

(continued)

DEFAULTS

The default line texture is a solid line.

NOTES

Since 7 bit positions are used, it is not possible to create a symmetric pattern.

When line-texturing is applied to a vector, the vector that is specified is displayed as a textured, rather than solid line. If the line is smaller than the pattern length, then as much of the pattern that can be displayed with the vector is displayed. If the line is smaller than the smallest element of the pattern, then the line is displayed as solid.

The **With Pattern** and **curve** commands create multiple vectors in memory. To the line-texturing hardware, each vector in a pattern or curve is seen as an individual vector. Line-texturing a patterned line or curve is the same as line-texturing a number of small segments. Curves and patterns affect line-texturing only in that they tend to create short vectors that may be too short to be completely textured.

NODE CREATED

This command creates a line texture operation node with line texture to be applied to all vectors below in the display structure hierachy. Sending a Boolean value to input <1> of the node turns the continuous texture feature on or off. Sending an integer value to the node changes the pattern.

Version A2.V01

FORMAT

```
name := VECtor_list [options] [N=n] vectors;
```

DESCRIPTION

Defines an object by specifying the points comprising the geometry of the object and their connectivity (topology).

PARAMETERS

name - Any legal PS 300 name.

options - Can be none, any, or all of the following five groups (but only one from each group, and in the order specified):

1. **BLOCK_normalized** - All vectors will be normalized to a single common exponent.

2. **COLOR** - This option is used when specifying color-blended vectors (refer to SET COLOR BLENDing command) to indicate that vector colors will be specified in lieu of vector intensities. When the **COLOR** option is used, the optional I=**i** clause used to specify the intensity of a vector (refer to the **vectors** parameter below) is replaced by the optional H=**hue** clause, where **H** is a number from 0 to 720 specifying the individual vector hues. The default is 0 (pure blue).

   The 0-720 scale for the H=**hue** clause is simply the SET COLOR scale of 0-360 repeated over the interval 360-720. On this scale, 0 represents pure blue, 120 pure red, 240 pure green, 360 pure blue again, 480 pure red again, 600 pure green again, and 720 pure blue. This "double color wheel" allows for color blending either clockwise or counterclockwise around the color wheel.

3. Connectivity:

   A. **CONNECTED_lines** - The first vector is an undisplayed position and the rest are endpoints of lines from the previous vector.

PARAMETERS (continued)

B. SEParate_lines - The vectors are paired as line endpoints.

C. DOTs - Each vector specifies a dot.

D. ITEMized - Each vector is individually specified as a move to position (P) or a line endpoint (L).

E. TABulated - This clause is used to specify an entry into a table that is used for specifying colors for raster lines and for specifying colors, radii, diffuse, and specular attributes for raster spheres. This option is also used to alter the attribute table itself.

When the TABulated option is used, the T=t clause replaces the I=i clause (for intensities) and the H=hue clause (for vector hues). The default is 127 (table entry 127).

There are 0 to 127 entries into the Attribute table. The Attribute table may be modified via input <14> of the SHADINGENVIRONMENT function.

4. Y and Z coordinate specifications (for constant or linearly changing Y and/or Z values):

    Y = y[DY=delta_y][Z = z[DZ=delta_z]]

where y and z are default constants or beginning values, and delta_y and delta_z are increment values for subsequent vectors.

5. INTERNAL_units - Vector values are in the internal PS 300 units [LENGTH]. Specifying this option speeds the processing of the vector list, but this also requires P/L information to be specified for each vector, and it doesn't allow default y values or specified intensities.

n - Estimated number of vectors.

PARAMETERS (continued)

vectors -The syntax for individual vectors will vary depending on the options specified in the **options** area. For all options except ITEMized, COLOR, and TABulated the syntax is:

```
xcomp[,ycomp[,zcomp]][I=i]
```

where xcomp, ycomp and zcomp are real or integer coordinates and i is a real number ($0.0 \leq i \leq 1.0$) specifying the intrinsic intensity for that point (1.0 = full intensity).

For ITEMized vector lists the syntax is:

```
P xcomp[,ycomp[,zcomp]][I=i]
```

or

```
L xcomp[,ycomp[,zcomp]][I=i]
```

where P means a move-to-position and L means a line endpoint.

If default y and z values are specified in the options area, they are not specified in the individual vectors.

For color-blended (COLOR) vector lists, the syntax is:

```
xcomp[,ycomp[,zcomp]][H=hue]
```

where xcomp, ycomp and zcomp are real or integer coordinates and hue is a real number between 0 and 720 specifying the hue of a vector.

For TABulated vector lists (TAB), the syntax is:

```
xcomp[,ycomp[,zcomp]][T=t]
```

where t is an integer between 0 and 127 specifying a table entry.

DEFAULTS

If not specified, the options default to:

1. Vector normalized
2. Not color blended
3. Connected
4. No default y or z values are assumed (see note 5)
5. Expecting internal units

Non color-blended vectors default to:

```
xcomp,ycomp[,zcomp][I=i]
```

If i is not specified, it defaults to 1.

Color-blended vectors default to:

```
xcomp,ycomp[,zcomp][H=hue]
```

If hue is not specified, it defaults to 0 (pure blue).

Tabulated vectors default to:

```
xcomp,ycomp[,zcomp][T=t]
```

If the table entry is not specified, it defaults to 127 (table entry 127).

NOTES

1. If n is less than the actual number of vectors, insufficient allocation of memory will result; if greater, more memory will be allocated than is used. (The former is generally the more severe problem.)

2. All vectors in a list must have the same number of components.

3. If y is specified in the options area, z must be specified in the options area.

NOTES (continued)

4. If no default is specified in the **options** area and no z components are specified in the **vectors** area, the vector list is a 2D vector list. If a z default is specified in the same case, the vector list is a 3D vector list.

5. The first vector must be a position (P) vector and will be forced to be a position vector if not.

6. Options must be specified in the order given.

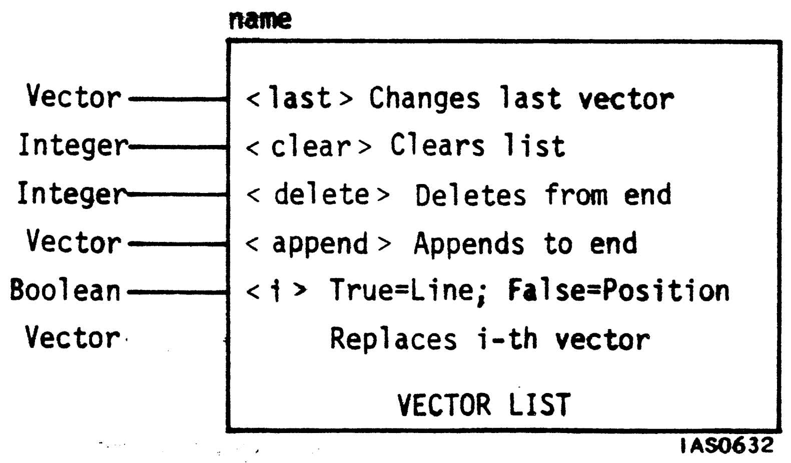
7. If CONNECTED_lines, SEParate_lines, or DOTs are specified in the options area but the vectors are entered using P/Ls, then the option specified takes precedence.

8. Block normalized vector lists generally take longer to process into the PS 300, but are processed faster for display once they are in the system.

DISPLAY TREE NODE CREATED

Vector list data node.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

1. Vector list nodes are in one of two forms:

   A. If DOTs was specified in the options area of the command, a DOT mode vector list node is created. The Boolean input to <i> is ignored in this case as well as the P/L portion of input vectors, and all vectors input are considered new positions for dots.

   B. All other vector list nodes created can be considered to be 2D or 3D ITEMized with intensity specifications after each vector, and if a 3D vector is input to a 2D vector list node, the last component modifies the intensity.

2. If a 2D vector is sent to a 3D vector list, the z value defaults to 0.

3. When you replace the i-th vector, the new vector is considered a line (L) vector unless it was first changed to a position vector with F:POSITION_LINE.

EXAMPLES

```
A := VECtor_list BLOCK SEParate INTERNAL N=4
     P 1,1 L -1,1 L -1,-1 L 1,-1;

B := VECtor_list n=5
     1,1 -1,1 I=.5
     -1,-1 1,-1 I=.75
     1,1;

C := VECtor_list ITEM N=5
     P 1,1
     L -1,1
     L -1,-1
     P 1,-1
     L 1,1;

D := VECtor_list TABulated N=5  {for drawing raster lines}
     P 0,1,0
     L 0,0,0  t=5
     L 1,0,0  t=2
     P 1,1,0  t=3
     L 0,1,0  t=4;
```

FORMAT

name := WRITEBACK [APPLied to name1];

DESCRIPTION

The WRITEBACK command creates a WRITEBACK operation node and delineates the data structure below the node for writeback operations. When the WRITEBACK operation node is activated, writeback is performed for name1.

PARAMETERS

name1 - The name of the structure or node to which writeback is applied.

NOTES

1. This node delimits the structure from which writeback data will be retrieved. Only the data nodes that are below the WRITEBACK operation node in the data structure will be transformed, clipped, viewport scaled, and sent back to the host.

2. Only a structure that is being displayed can be enabled for writeback. This means that the WRITEBACK operation node must be traversed by the display processor and so must be included in the displayed portion of the structure. If the writeback of only a portion of the picture is desired, WRITEBACK nodes must be placed appropriately in the display structure.

3. Any number of WRITEBACK nodes can be placed within a structure. Only one writeback operation can occur at a time. If more than one node is triggered, the writeback operations are performed in the order in which the corresponding nodes were triggered. If the user creates any WRITEBACK nodes (other than the WRITEBACK node created initially at boot-up), these nodes must be displayed before being triggered. If the nodes are triggered before being displayed, an error message will result.

4. The terminal emulator and message_display data will not be returned to the host.

DISPLAY TREE NODE CREATED

The command creates a WRITEBACK operation node.

Table 1. Key to Abbreviations for Valid Data Types

KEY TO VALID DATA TYPES	
Any	Any message
B	Boolean value
C	Constant value
CH	Character
I	Integer
Label	Data input to LABELS node
M	2x2, 3x3, 4x3, 4x4 matrix
PL	Pick list
R	Real number
S	Any string
Special	Special data type
V	Any vector
2D	2D vector
3D	3D vector
4D	4D vector
2x2	2x2 matrix
3x3	3x3 matrix
4x3	4x3 matrix
4x4	4x4 matrix

Conjunctive/Disjunctive Sets

Some PS 300 functions have conjunctive or disjunctive inputs and outputs. A function with conjunctive inputs must have a new message on every input before it will activate. A function with conjunctive outputs will send a message on every output when the function is activated.

Conversely, a disjunctive-input function does not require a new message on every input to activate. A disjunctive-output function may not send a message on each output (or any output) every time it receives a complete set of input messages.

The F:ADD function, for example, has conjunctive inputs. A value must be sent to each of the two inputs before the function will fire. The inputs are then added together, which produces an output that is the sum of the inputs. The output is conjunctive. Unlike F:ADD, F:ADDC is a disjunctive-input function; it does not require a new message on every input.

F:BROUTE, on the other hand, is a conjunctive-input, disjunctive-output function. Both inputs require messages to activate the function. However, a message will be sent out only one of the outputs, depending on the value received on input 1.

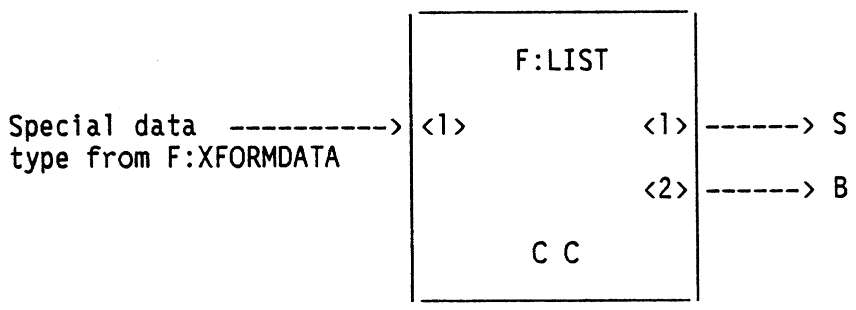
F:ACCUMULATE is an example of different sort of disjunctive output. Every input does not produce an output. The function activates each time a new message is received on input 1, but the output fires at specified intervals rather than each time the function is activated.

The following notation is used in the Function Summary to indicate conjunctive and disjunctive inputs and outputs.

KEY TO CONJUNCTIVE/DISJUNCTIVE SYMBOLS

CC	conjunctive inputs, conjunctive outputs
CD	conjunctive inputs, disjunctive outputs
DC	disjunctive inputs, conjunctive outputs
DD	disjunctive inputs, disjunctive outputs

Version A1.V02



PURPOSE

Converts the output of the F:XFORMDATA function to an ASCII string. This function is always used with F:XFORMDATA.

DESCRIPTION

INPUT
<1> – data output by F:XFORMDATA

OUTPUT
<1> – resulting ASCII string
<2> – Boolean (TRUE)

DEFAULTS

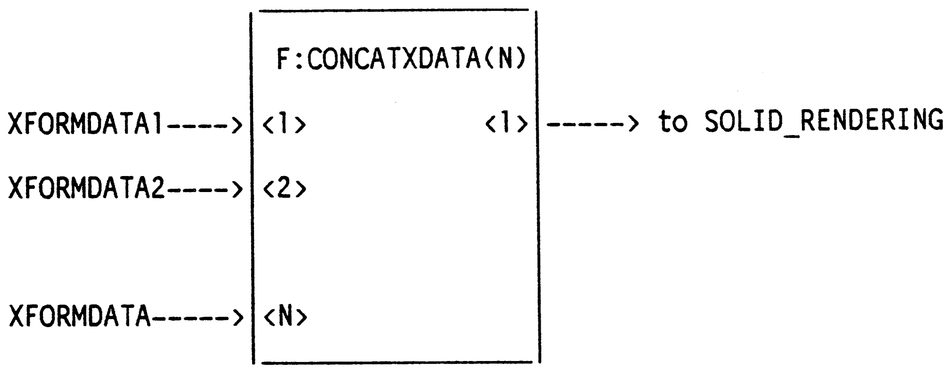
None.

NOTES

1. Input <1> is always connected to output <1> of F:XFORMDATA.

2. Output <2> is TRUE when processing is complete. There is no output otherwise.

3. Output <2> should be connected to an instance of F:SYNC(2) to synchronize F:LIST completion with the initiation of a subsequent transformed-data request.



PURPOSE

Accepts up to 127 transformed vector lists (output from XFORMDATA functions) and concatenates them into a single transformed vector list.

DESCRIPTION

INPUT
<1> – output of F:TRANSFORMDATA (transformed vector list)
•
•
•
<N> – output of F:TRANSFORMDATA (transformed vector list)

OUTPUT
<1> – concatenated vector list

NOTES

1. This function is used to avoid the maximum vector restriction imposed on the output of F:XFORMDATA. The XFORMDATA function will return a maximum of 2048 vectors. To obtain a rendering on the PS 340 raster display of greater than 2048 vectors, the output of multiple instances of XFORMDATA must be concatenated into a single transformed vector list which can be sent to the rendering node.

2. Inputs <1> through <N> accept a transformed vector list output from F:XFORMDATA.

```
                 ___________________
                |                   |
                |    F:PICKINFO     |
                |                   |
PL ------>      |<1>             <1>|----> I
                |                   |
I ------->      |<2> C           <2>|----> S
                |                   |
                |                <3>|----> 2D, 3D
                |                   |
                |                <4>|----> I
                |                   |
                |                <5>|----> B
                |                   |
                |                <6>|----> R
                |                   |
                |                <7>|----> I
                |                   |
                |                <8>|----> Special
                |                   |
                |                <9>|----> 2D
                |        D D        |
                |___________________|
```

PURPOSE

Reformats picklist information for use by other functions. The output picklist is separated into its component parts.

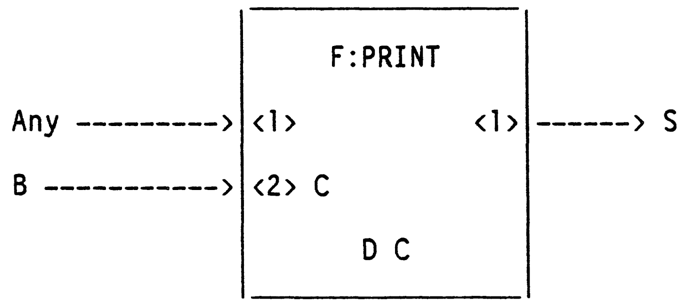
DESCRIPTION

INPUT
<1> – picklist
<2> – depth within structure reported (constant)

OUTPUT
<1> – index
<2> – pick identifier(s)
<3> – coordinates
<4> – dimension
<5> – coordinates reported
<6> – curve parameter, t
<7> – data type code
<8> – name of picked element
<9> – screen coordinates of the picked point

PS 350 Modifications

Output <9> has been added to F:PICKINFO. This output reports the screen coordinates of a pick.



PURPOSE

Converts any data type to string format; that is, it performs an inverse of the operation that occurs when an ASCII string is input to the PS 300 and is converted to one of the data types.

DESCRIPTION

INPUT
<1> – any message
<2> – Boolean governing numeric format (constant)

OUTPUT
<1> – string

PS 350 Modification

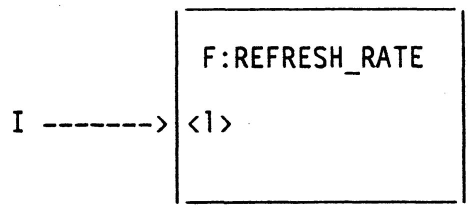
Screen coordinates, if passed to the function from F:PICKINFO, are added to the string output on <1>. Output <1> has been modified to report a pick in which coordinate picking information is given:

For a vector declared in a VECTOR_LIST, the output string format is:

<1><dimension><pick_x><pick_y>[<pick_z>]<t>
<pick ID's><screen_x><screen_y>

For a vector within a polynomial curve the output string format is:

<2><dimension><pick_x><pick_y>[<pick_z>]<t>
<pick ID's><screen_x><screen_y>

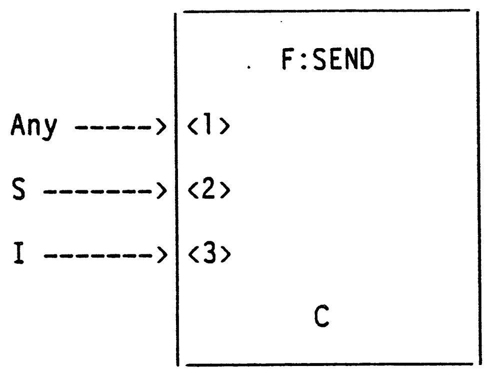


PURPOSE

Locks refresh rate. This function accepts an integer on input <1>. The integer must be in the range of 2 through 5. This is the number of ticks per refresh frame (ticks occur at twice the line frequency). The actual refresh rate depends on the line frequency.

Ticks	60Hz	50Hz
2	60	50
3	40	33
4	30	25
5	24	20

Version A1.V02



PURPOSE

This is the function network equivalent of the SEND command. It allows you to send any valid data type to any named entity at any valid index.

DESCRIPTION

INPUT

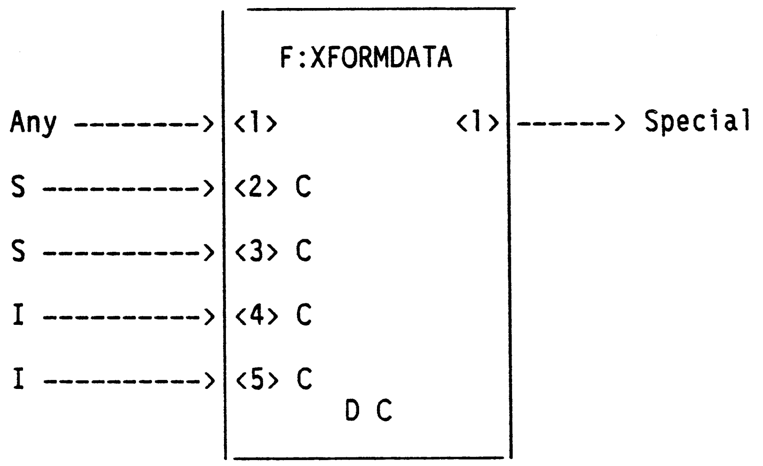
<1> - message sent
<2> - name of the destination node
<3> - index into the destination node

NOTES

1. This function has no output.

2. Input <1> accepts special data types that most functions do not accept, such as the data type output by F:LABEL.

3. The SETUP CNESS command can be used to specify constant inputs as default values.



PURPOSE

Sends transformed data (either a vector list or a 4x4 matrix) to a specified destination (e.g., the host, a printer, or the screen).

DESCRIPTION

INPUT
<1> - any message
<2> - name of XFORM node (constant)
<3> - name of destination object (constant)
<4> - destination vector index (constant)
<5> - number of vectors (constant)

OUTPUT
<1> - special data type used exclusively as input to F:LIST

DEFAULTS

Default for input <4> is 1, default for input <5> is 2048.

NOTES

1. Input <1> is a trigger for F:XFORMDATA. This input would typically be connected to a function button, either directly or via F:SYNC(2), allowing transformed data to be requested easily.

2. Input <2> is a string or matrix containing the name of the XFORM command in the display tree (either XFORM MATRIX or XFORM VECtor). By referring to an XFORM command, this input indirectly specifies the object whose transformed data is to be sent. If the string names something other than an XFORM command, an error message is displayed. If the string names a node which does not exist, an error message is sent and the message is removed from input <2>.

3. Input <3> is a string containing the name to be associated with the transformed vectors. The name need not be previously defined. If this input does not contain a valid string, the transformed matrix or vectors will be created without a name (an acceptable situation unless the transformed vectors need to be referenced or displayed.) The transformed vector list can be displayed or modified, provided a name is given on this input. The transformation matrix cannot be used, however, so naming and sending it to input <3> is not useful.

4. Input <4> is an integer index specifying the place in a vector list at which the PS 300 is to start returning transformed data. This input is only used when the command name at input <2> represents an XFORM VECtor command (not an XFORM MATRIX command). The default value is 1.

5. Input <5> is an integer number of consecutive vectors for which transformed data is to be returned, starting at the vector specified at input <4>. This input is only used when the command name at input <2> represents an XFORM VECtor command (not an XFORM MATRIX command). No more than 2048 consecutive vectors may be returned. The default value is 2048.

6. Output <1> contains the transformed data in a format which can only be accepted by input <1> of F:LIST. (F:LIST then prints out the data in ASCII format -- either a PS 300 VECTOR_LIST command or a PS 300 MATRIX_4X4 command, depending on whether the command named at input <2> was an XFORM VECtor or an XFORM MATRIX.)

7. F:XFORMDATA is used in connection with rendering lines and spheres on the PS 340 raster display. This functionality is described in Version A2.V01 of the PS 340 Graphics Firmware Release Notes.

```
                      CSM
                     (CSM2)
               ________________
              |                |
   B ------>  |<1>          <1>| ------> Connected to System
              |                |         at initialization
              |       C C      |
              |________________|
```

PURPOSE

Sets the Color Shadow Mask (CSM) calligraphic display on or off for the Terminal Emulator, for MESSAGE_DISPLAY and for the user's data structures.

DESCRIPTION

INPUT
<1> – TRUE = CSM on, FALSE = CSM off

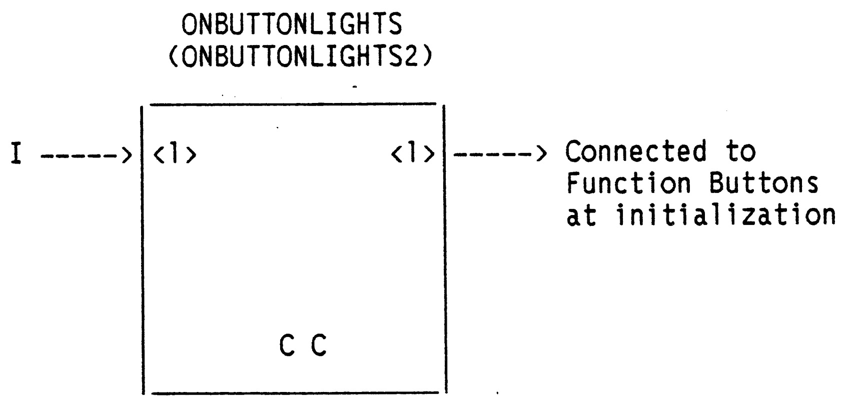
OUTPUT
<1> – connected to System

DEFAULT

The default is FALSE, setting the CSM off.

NOTES

1. A TRUE sent to input <1> of CSM slows the speed of the line generator for the CSM calligraphic display. This results in lines that have brighter colors and better end point match.



PURPOSE

Turns on lighted buttons on the Function Buttons unit.

DESCRIPTION

INPUT
<1> - integer (1 through 32) indicating the button number

OUTPUT
<1> - connected to Function Buttons

NOTES

1. Each button may be turned on independently or all buttons may be turned on by a single message. A zero (0) or any out-of-range integer at input <1> turns on all button lights. An integer from 1 to 32 at input <1> turns on the corresponding button light.

2. Function buttons are arranged in one row of four, four rows of six, and another row of four. They are numbered from left to right starting from the top row. The top row is numbered 1 through 4; the second row 5 through 10, and so on until the last row, 29 through 32.

```
                       PICK
                      (PICK2)
                 ____________________
                |                    |
Any ----------->|<1>              <1>|------> PL
                |                    |
B ------------->|<2> C            <2>|------> B
                |                    |
I ------------->|<3> C            <3>|------> B
                |                    |
                |        D D         |
                |____________________|
```

PURPOSE

Interfaces with the hardware picking circuitry. Any message on input <1> arms the PICK function. Once PICK is enabled, when a pick occurs, the pick list associated with the picked data is sent out on output <1> and a Boolean FALSE is sent on output <2>. Typically, this Boolean is used to disable picking of a set of objects by connecting it to a SET PICKING ON/OFF node in a display tree.

DESCRIPTION

INPUT
<1> – trigger
<2> – TRUE = coordinate, FALSE = index (constant)
<3> – timeout duration (constant)

OUTPUT
<1> – pick list
<2> – FALSE = pick enabled
<3> – FALSE = ACP attempted an unsuccessful pick or timeout occurred

PS 350 Modification

As noted above, output <3> of PICK now reports a FALSE when the ACP attempts a pick and is unsuccessful as well as when the timeout specified on input <3> is exceeded.

```
                         PICK
                        (PICK2)
                  ___________________
                 |                   |
Any ----------->|<1>             <1>|------> PL
                 |                   |
B ------------->|<2> C           <2>|------> B
                 |                   |
I ------------->|<3> C           <3>|------> B
                 |                   |
R ------------->|<4>                 |
                 |                   |
I ------------->|<5>                 |
                 |                   |
R ------------->|<6>                 |
                 |                   |
                 |        D D        |
                 |___________________|
```

PURPOSE

Interfaces with the hardware picking circuitry. Any message on input <1> arms the PICK function. Once PICK is enabled, when a pick occurs, the pick list associated with the picked data is sent out on output <1> and a Boolean FALSE is sent out on output <2>. Typically, this Boolean is used to disable picking of a set of objects by connecting it to a SET PICKING ON/OFF node in a display tree.

DESCRIPTION

INPUT
- <1> - trigger
- <2> - TRUE = coordinate, FALSE = index (constant)
- <3> - timeout duration (constant)
- <4> - defines pick window half size for the ACP pass of the pick
- <5> - retry count
- <6> - half-size increment to be added to window half-size on each retry

OUTPUT
- <1> - pick list
- <2> - FALSE = pick enabled
- <3> - FALSE = timeout elapsed

PS 350/PS 390
A2.V02 - April 1987(continued)

NOTES

1. Input <2> selects the kind of pick list that will be output on output <1>. A FALSE on input <2> indicates that the output pick list will be the pick identifier and an index into the vector list or the character string. (The index into the vector list identifies its position in the list; vector 3 is the third vector in a vector list. The index into a character string identifies the picked character by its position in the string; character 5 is the fifth character in a string.)

2. A TRUE on input <2> indicates that the output pick list will include, in addition to the pick identifier and the index, the picked coordinates and the dimension of the picked vector. If the vector is part of a polynomial curve, its parameter value, t, is supplied instead of the index.

3. Coordinate picking on a character string returns an index into the string, not its picked coordinates.

4. Coordinate picking cannot be performed on a vector over 500 [LENGTH] units long.

5. The pick list on output <1> is typically connected to an instance of F:PICKINFO to convert the pick list to a locally useful format. If the pick list is to be printed out, output <1> may be connected to F:PRINT to convert the pick list code to printable characters.

6. When several vectors are picked, the first vector drawn by the Line Generator is reported as picked. For example, if three vectors in a single vector list were picked simultaneously (at a point of intersection), the first vector listed in the object definition would be reported as picked.

7. The integer on input <3> specifies a pick timeout period in refresh frames. This pick timeout period allows the user to determine whether a pick has occurred within the specified amount of time. Timing starts when the PICK function is armed with a message on active input <1>. Allowable integers for input <3> are from 4 through 60.

NOTES (continued)

8. If input <3> is not used, all picks will be reported once the function is armed because no timeout duration has been specified.

9. Typically, the FALSE at output <3> would be used to turn off picking in a display tree (at a SET PICKING ON/OFF node) or to send a "NO PICK" message (probably via F:SYNC(2)) back to the host.

10. The user has three means of cancelling an existing pick timeout duration:

    a. Send an INITialize command. This will remove the PICK function and replace it with a new instance of the PICK function.

    b. Send a non-integer (and ignore the "Bad message" error).

    c. Send an integer less than 4 or greater than 60 to input <3> (and ignore the "Bad message" error).

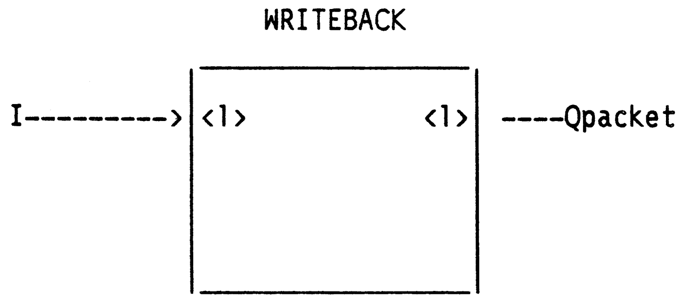
11. Input <4> is a real number between 0 and 1 that defines the pick window half-size for the ACP pass of the pick. This is different from the size set by the SET PICKing LOCation operation node. The Line Generator or the Frame Buffer uses the operation node to determine if a pick has occurred; whereas the ACP uses input <4> to do the actual pick pass on the data.

12. Input <5> is an integer specifying pick pass retries. Since it is possible that the ACP will not find the picked data during a pick pass, input <5> indicates the number of times to add the window increment on input <6> and try another pick pass.

13. Input <6> is a real number between 0 and 1 which specifies the amount to increase the pick window half size on each retry of the pick pass.

EXAMPLE

If a 10 is sent to constant input <3>, then the PICK function is armed with a message on input <1>. The function waits 10 refresh frames from the time the input <1> message is received before checking to see if a pick has occurred. If a pick has occurred within that period, the function outputs the appropriate pick list. If a pick has not occurred, the function outputs a FALSE on output <3>. In either case, the PICK function is disarmed and must be rearmed via input <1> before further picking can be reported.



PURPOSE

WRITEBACK is initialized by the system and is used to send encoded writeback data to user function networks.

This function is not activated by the normal input queue triggering mechanism. It is activated by sending a TRUE to any WRITEBACK operation node.

DESCRIPTION

INPUT

WRITEBACK has one input queue. Input <1> accepts integers specifying the size of Qpackets to be output by the function. The default size is 512. Minimum and maximum sizes are 16 and 1024. If the size specified on the input is not within this range, the default size will be used.

OUTPUT

WRITEBACK has one output queue. Output <1> passes the encoded writeback data out as Qpackets.

NOTES

WRITEBACK will return all data that are under the WRITEBACK operation node. Host-resident code will be responsible for recognizing the start-of-writeback and end-of-writeback commands. Attribute information, such as color, must be interpreted by host code to ensure that the hardcopy plots are correct.

On the PS 350, viewport translations have not been applied to the data. To correctly compute the position of endpoints, the host program interpreting the writeback code must add a viewport center to each endpoint. The initial viewport center is established with a VIEWPORT CENTER command. The VIEWPORT CENTER command is sent following the start-of-writeback command. Any changes to the viewport center will be indicated through this sequence of commands: CLEAR DDA, CLEAR SAVE POINT, position endpoint, CLEAR SAVE POINT. The position endpoint becomes the new viewport center.

UTILITY PROCEDURE AND PARAMETERS

```
PROCEDURE PAttach (   %DESCR Modifiers : P_VaryingType;
                      PROCEDURE Error_Handler (Error : INTEGER));
```

DEFINITION

This procedure attaches the PS 300 to the communications channel.

If this procedure is not called prior to use of the Application Procedures, the error code value corresponding to the name PSE_NotAtt is generated, indicating that the PS 300 communications link has not been established.

The parameter (Modify) must contain the phrases:

```
LOGDEVNAM=name/PHYDEVTYP=type
```

where "name" refers to the logical name of the device that the GSRs will communicate with, i.e. TTA6:, TTB2: XME0:, PS:, etc. and "type" refers to the physical device type of the hardware interface that the GSRs will communicate through. This last argument can only be one of the following four interfaces:

ASYNC (standard RS-232 asynchronous communication interface)
PARALLEL (Parallel interface option)
ETHERNET (DECnet Ethernet option)

The parameter string must contain EXACTLY one "/" and blanks are NOT allowed to surround the "=" in the phrases. The PAttach parameter string is not sensitive to upper or lower case.

Example: PAttach ('logdevnam=tta2:/phydevtyp=async', Error_Handler);

where "tta2" is the logical device name of the PS 300, and the hardware interface is standard asynchronous RS-232.

Example: PAttach ('logdevnam=ps:/phydevtyp=dmr-11', Error_Handler);

where the physical device type is a DMR-11 interface, and where the user has informed the VAX that the logical symbol "ps" refers to the name of the logical device that the GSRs will communicate with using the following ASSIGN command:

```
$ ASSIGN XMD0: PS
$ RUN <application-pgm>
```

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PAttch (Modify, ErrHnd)
```

where:

Modify is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine attaches the PS 300 to the communications channel. If this subroutine is not called prior to use of the Application Subroutines, the user's error handler is invoked with the "The PS 300 communications link has not been established" error code corresponding to the mnemonic: PSENOA:.

The parameter (Modify) must contain the phrases:

LOGDEVNAM=name/PHYDEVTYP=type

where "name" refers to the logical name of the device that the GSRs will communicate with, i.e. TTA6:, TTB2: XMEO:, PS:, etc. and "type" refers to the physical device type of the hardware interface that the GSRs will communicate through. This last argument can only be one of the following four interfaces:

ASYNC (standard RS-232 asynchronous communication interface)
PARALLEL (high speed parallel interface
ETHERNET (DECnet Ethernet option)

The parameter string must contain EXACTLY 1 "/" and blanks are NOT allowed to surround the "=" in the phrases. The Pattch parameter string is not sensitive to upper or lower case.

Example: CALL PAttch ('logdevnam=tta2:/phydevtyp=async', Errhnd)

where "tta2" is the logical device name of the PS 300, and the hardware interface is standard asynchronous RS-232.

(Continued on next page)

```
     Example:  CALL PAttch ('logdevnam=ps:/phydevtyp=dmr-11', ErrHnd)
```

where the physical device type is a DMR-11 interface and where the user has informed the VAX that the logical symbol "ps" refers to the name of the logical device that the GSRs will communicate with using the following ASSIGN command:

```
     $ ASSIGN XMD0: PS:
     $ RUN <application-pgm>
```

UTILITY PROCEDURE AND PARAMETERS

```
[GLOBAL] PROCEDURE PDevInfo (    VAR Channel_num     : INTEGER;
                                 VAR Device_type     : INTEGER;
                                 VAR Dev_status      : INTEGER;
                             PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure is used to return the Q I/O channel number so that users do not need to detach from the GSRs while doing Physical I/O.

Channel is the VAX Q I/O channel number.

Device is the device code, where:

1 is the code for the DRM-11 interface
2 is the code for the standard asynchronous interface
3 is the code for the Parallel interface

Status is the status where:

0 is not attached
1 is attached

Version A1.V02 - March 1985

UTILITY SUBROUTINE AND PARAMETERS

```
CALL PDINFO (Channel, Device, Status, ErrHnd)
```

where:

Channel is an INTEGER*4 that is the VAX Q I/O channel number

Device is an INTEGER*4 that is the device code, where:

1 is the code for the DRM-11 interface
2 is the code for the asynchronous interface
3 is the code for the Parallel interface

Status is an INTEGER*4 that is the status where:

0 is not attached
1 is attached

ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine is used to return the Q I/O channel number so that users do not need to detach from the GSRs while doing Physical I/O.

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PVcBeg (Name, VecCou, BNorm, CBlend, Dimen, Class, ErrHnd)
```

where:

Name is a CHARACTER STRING defining the name of the vector list

VecCou is an INTEGER*4 specifying the total number of vectors in the vector list

BNorm is a LOGICAL*1 defined: .TRUE. for Block Normalized, .FALSE. for Vector Normalized

CBlend is a LOGICAL*1 defined: .TRUE. for Color Blending, .FALSE. for normal depth cueing

Dimen is an INTEGER*4 2 or 3 (2 or 3 dimensions respectively)

*Class is an INTEGER*4 defining the class of the vector list

ErrHnd is the user-defined error-handler subroutine.

This subroutine must be called to begin a vector list. To send a vector list, the user must call:

PVcBeg

PVcLis (This may be called multiple times for vector-normalized vector lists.)

PVcEnd

Together, the above 3 subroutines implement the PS 300 command:

Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

NOTE

The dimension must be specified in the PVCBEG application subroutine. In the PS 300 command, dimension is implied by syntax.

(Continued on next page)

* These mnemonics may be referenced directly by the user if PROCONST.FOR is INCLUDED in the subroutine. See the section on Programming Suggestions for a description of PROCONST.FOR. A description of the vector classes and their INTEGER*4 value is given below.

Mnemonic	Meaning	INTEGER*4 Value
PVCONN	Connected	0
PVDOTS	Dots	1
PVITEM	Itemized	2
PVSEPA	Separate	3
PVTAB	Tabulated	4

Note: If the vector list is class PVTAB, then the BNorm must be FALSE and Dimen must be equal to 3; that is, tabulated vector lists must be vector-normalized 3D vector lists.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PVcBeg (Name, VecCou, BNorm, CBlend, Dimen, Class, ErrHnd)
```

where:

Name is a CHARACTER STRING defining the name of the vector list

VecCou is an INTEGER*4 specifying the total number of vectors in the vector list

BNorm is a LOGICAL*1 defined: .TRUE. for Block Normalized, .FALSE. for Vector Normalized

CBlend is a LOGICAL*1 defined: .TRUE. for Color Blending, .FALSE. for normal depth cueing

Dimen is an INTEGER*4 2 or 3 (2 or 3 dimensions respectively)

*Class is an INTEGER*4 defining the class of the vector list

ErrHnd is the user-defined error-handler subroutine.

This subroutine must be called to begin a vector list. To send a vector list, the user must call:

PVcBeg

PVcLis (This may be called multiple times for vector-normalized vector lists)
PVcEnd

Together, the above 3 subroutines implement the PS 300 command:

Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

NOTE

The dimension must be specified in the PVCBEG application subroutine. In the PS 300 command, dimension is implied by syntax.

(Continued on next page)

* These mnemonics may be referenced directly by the user if PROCONST.FOR is INCLUDED in the subroutine. See the section on Programming Suggestions for a description of PROCONST.FOR. A description of the vector classes and their INTEGER*4 value is given below.

Mnemonic	Meaning	INTEGER*4 Value
PVCONN	Connected	0
PVDOTS	Dots	1
PVITEM	Itemized	2
PVSEPA	Separate	3
PVTAB	Tabulated	4

Note: If the vector list is class PVTAB, then the BNorm must be FALSE and Dimen must be equal to 3; that is, tabulated vector lists must be vector normalized 3D vector lists.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

CALL PVcLis (NVec, Vecs, PosLin, ErrHnd)

where:

NVec is the number of vectors in the vector list and is defined: INTEGER*4

Vecs is the array containing the vectors of the vector list and is defined: REAL*4 (4, NVec)

```
        where:  Vecs(1,n) = vector n x-component
                Vecs(2,n) = vector n y-component
                Vecs(3,n) = vector n z-component
                Vecs(4,n) = vector n intensity (or hue)
                     0 <= Vecs(4,n) <=1  or
                     0 <= Vecs(4,n) <=127 if vector
                        class is tabulated.
```

PosLin is the array containing the move/positive - draw/line information for each vector. PosLin is defined : LOGICAL*1 PosLin(NVec)

If PosLin(n) = .TRUE. then vector n is a draw(line) vector.

If PosLin(n) = .FALSE. then vector n is a move(position) vector.

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine must be called to send a piece of a vector list. For vector-normalized vector lists, this subroutine can be called multiple times to send the vector list down in pieces. Multiple calls to this subroutine are not permitted for the block-normalized vector list case, unless the subroutine PVcMax is called first. To send a vector list, the user must call:

PVcBeg

PVcLis (This may be called multiple times for vector-normalized vector lists)

PVcEnd

(Continued on next page)

The POSLIN Array is always required, however the CLASS specified in PVcBeg determines how it is used. For CONNECTED, DOTS, and SEPARATE, the user need not specify the contents of POSLIN. For ITEMIZED and TABULATED, the user-specified position/line is used.

The fourth position of Vecs is the intensity of that vector if vector-normalized, regardless of dimension. If block-normalized, the first vector's fourth position is used as the entire vector list intensity.

The fourth position of Vecs can be used to specify color in lieu of intensity when specifying color-blended vectors (refer to PSETCB). Use the following algorithm to convert the acceptable range of hues (real numbers 0-720 for the PS 300 VECTOR_LIST command) to the expected range of 0-1 for the PVCLIS GSR routine before sending.

- If the value is less than 0 or greater than 720, clamp it to the nearest in-range value.
- If the value is greater than or equal to 360, subtract 360.
- Divide the value by 768.
- If the original value was greater than or equal to 360, add .5 to the result of the division.

This has the effect of mapping hue values in the range (0-360) to (0-.46875), and values in the range (360-720) to (.5-.96875). Values greater than .46875 and less than .5 are out of range, and are interpreted as .5 (pure blue).

If the vector class is "tabulated," the fourth position of the VECS is an INDEX. Users should specify whole numbers $0 \leq$ index ≤ 127 in this case. The GSRs will truncate the value supplied to an integer and force the value to be in range 0 to 127.

Together, the subroutines PVcBeg, PVcLis, and PVcEnd implement the PS 300 command:

```
Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE,
        TABULATED) N=n <vectors>;
```

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PVcLis (NVec, Vecs, PosLin, ErrHnd)
```

where:

NVec is the number of vectors in the vector list and is defined: INTEGER*4

Vecs is the array containing the vectors of the vector list and is defined: REAL*4 (4, NVec)

```
          where:  Vecs(1,n) = vector n x-component
                  Vecs(2,n) = vector n y-component
                  Vecs(3,n) = vector n z-component
                  Vecs(4,n) = vector n intensity (or hue)
                      0 <= Vecs(4,n) <=1  or
                      0 <= Vecs(4,n) <=127 if vector
                         class is tabulated.
```

PosLin is the array containing the move/positive – draw/line information for each vector. PosLin is defined : LOGICAL*1 PosLin(NVec)

If PosLin(n) = .TRUE. then vector n is a draw(line) vector.

If PosLin(n) = .FALSE. then vector n is a move(position) vector.

ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine must be called to send a piece of a vector list. For vector-normalized vector lists, this subroutine can be called multiple times to send the vector list down in pieces. Multiple calls to this subroutine are not permitted for the block-normalized vector list case, unless the subroutine PVcMax is called first. To send a vector list, the user must call:

PVcBeg

PVcLis (This may be called multiple times for vector normalized vector lists)

PVcEnd

(Continued on next page)

The POSLIN Array is always required, however the CLASS specified in PVcBeg determines how it is used. For CONNECTED, DOTS, and SEPARATE, the user need not specify the contents of POSLIN. For ITEMIZED and TABULATED, the user-specified position/line is used.

The fourth position of Vecs is the intensity of that vector if vector-normalized, regardless of dimension. If block-normalized, the first vector's fourth position is used as the entire vector list intensity.

The fourth position of Vecs can be used to specify color in lieu of intensity when specifying color-blended vectors (refer to PSETCB). Use the following algorithm to convert the acceptable range of hues (real numbers 0-720 for the PS 300 VECTOR_LIST command) to the expected range of 0-1 for the PVCLIS GSR routine before sending.

- If the value is less than 0 or greater than 720, clamp it to the nearest in-range value.
- If the value is greater than or equal to 360, subtract 360.
- Divide the value by 768.
- If the original value was greater than or equal to 360, add .5 to the result of the division.

This has the effect of mapping hue values in the range (0-360) to (0-.46875), and values in the range (360-720) to (.5-.96875). Values greater than .46875 and less than .5 are out of range, and are interpreted as .5 (pure blue).

If the vector class is "tabulated," the fourth position of the VECS is an INDEX. Users should specify whole numbers $0 \leq$ index ≤ 127 in this case. The GSRs will truncate the value supplied to an integer and force the value to be in range 0 to 127.

Together, the subroutines PVcBeg, PVcLis, and PVcEnd implement the PS 300 command:

```
Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE,
        TABULATED) N=n <vectors>;
```

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
SUBROUTINE PVCMAX  (MAX, ERRHAND)
```

DEFINITION

This subroutine must be called before calling PVCLis if creating a creating a block-normalized vector list with multiple calls to PVCLis. To send a vector list, the user must call:

- PVCBeg
- PVCMax (If making calls to PVCLis and creating a block-normalized vector list.)
- PVCLis (This may be called multiple times for vector-normalized vector lists.)
- PVcEnd (This must be last.)

Together, the above 4 procedures implement the PS 300 command

```
Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE)
        N=n <vectors>;
```

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
SUBROUTINE PVCMAX  (MAX, ERRHAND)
```

DEFINITION

This subroutine must be called before calling PVCLis if creating a creating a block-normalized vector list with multiple calls to PVCLis. To send a vector list, the user must call:

- PVCBeg
- PVCMax (If making calls to PVCLis and creating a block-normalized vector list.)
- PVCLis (This may be called multiple times for vector-normalized vector lists.)
- PVcEnd (This must be last.)

Together, the above 4 procedures implement the PS 300 command

```
Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE)
        N=n <vectors>;
```

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
     PROCEDURE PVecBegn (   %DESCR Name               : P_VaryingType;
                                   VectorCount        : INTEGER;
                                   BlockNormalized    : BOOLEAN;
                                   ColorBlending      : BOOLEAN;
                                   Dimen              : INTEGER;
                                   Class              : INTEGER;
                         PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to begin a vector list. To send a vector list, the user must call the procedures:

PVecBegn

PVecList (This procedure may be called multiple times for vector-normalized vector lists)

PVecEnd

It contains the following parametric definitions:

- Name specifies the name to be given to the vector list
- VectorCount is the number of vectors to be created
- BlockNormalized is TRUE for Block Normalized and FALSE for Vector Normalized
- ColorBlending is TRUE for Color Blending and FALSE for normal depth cueing
- Dimen is 2 or 3 (2 or 3 dimensions respectively)
- *Class corresponds to a vector class
- Error_Handler is the user-defined error-handler procedure

(Continued on next page)

Together, the above 3 procedures implement the PS 300 command:

Name := VECTOR__LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

NOTE

The dimension must be specified in the PVECBEGN application procedure. In the PS 300 command, dimension is implied by syntax.

* These mnemonics may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure.

Mnemonic	Meaning	INTEGER Value
P_Conn	Connected	0
P_Dots	Dots	1
P_Item	Itemized	2
P_Sepa	Separate	3
P_Tab	Tabulated	4

Note: If the vector list is class P_Tab, BlockNormalized must be FALSE, and Dimen must be equal to 3; that is, tabulated vector lists must be vector-normalized 3D vector lists.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PVecBegn (   %DESCR Name              : P_VaryingType;
                              VectorCount       : INTEGER;
                              BlockNormalized   : BOOLEAN;
                              ColorBlending     : BOOLEAN;
                              Dimen             : INTEGER;
                              Class             : INTEGER;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to begin a vector list. To send a vector list, the user must call the procedures:

PVecBegn

PVecList (This procedure may be called multiple times for vector-normalized vector lists)

PVecEnd

It contains the following parametric definitions:

- Name specifies the name to be given to the vector list
- VectorCount is the number of vectors to be created
- BlockNormalized is TRUE for Block Normalized and FALSE for Vector Normalized
- ColorBlending is TRUE for Color Blending and FALSE for normal depth cueing
- Dimen is 2 or 3 (2 or 3 dimensions respectively)
- *Class corresponds to a vector class
- Error_Handler is the user-defined error-handler procedure

(Continued on next page)

Together, the above 3 procedures implement the PS 300 command:

Name := VECTOR__LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

NOTE

The dimension must be specified in the PVECBEGN application procedure. In the PS 300 command, dimension is implied by syntax.

* These mnemonics may be referenced directly by the user if PROCONST.PAS is INCLUDED in the procedure.

Mnemonic	Meaning	INTEGER Value
P_Conn	Connected	0
P_Dots	Dots	1
P_Item	Itemized	2
P_Sepa	Separate	3
P_Tab	Tabulated	4

Note: If the vector list is class P_Tab, BlockNormalized must be FALSE, and Dimen must be equal to 3; that is, tabulated vector lists must be vector-normalized 3D vector lists.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PVecList (     NumberOfVectors   : INTEGER;
                         VAR Vectors       : P_VectorListType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to send a piece of a vector list. For vector-normalized vector lists, this procedure can be called repeatedly to send the vector list down in pieces. Multiple calls to this procedure are not permitted for the block-normalized vector list case, unless the procedure PVecMax is called first. To send a vector list, the user must call the procedures:

PVecBegn

PVecList (This procedures may be called multiple times for vector-normalized vector lists)

PVecEnd

Together, the above 3 procedures implement the PS 300 command:

Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

Vectors is the array containing the vectors of the vector list.

where: Vectors [n].V4[1] := Vector n x-component
Vectors [n].V4[2] := Vector n y-component
Vectors [n].V4[3] := Vector n z-component
Vectors [n].V4[4] := Vector n intensity (or hue)
0 <= vectors [n].V4[4] <=1 or 0<= Vectors[n].V4[4] <=127 if vector class is tabulated.

Vectors [n].Draw := True if vector n is a draw/line vector.
Vectors [n].Draw := False if vector n is a move/position vector.

The fourth position of Vectors is the intensity of that vector if vector-normalized, regardless of dimension. If block-normalized, the first vector's fourth position is used as the entire vector list intensity.

The fourth position of Vectors can be used to specify color in lieu of intensity when specifying color-blended vectors (refer to PSETBLND). Use the following algorithm to convert the acceptable range of hues (real numbers 0–720 for the PS 300 VECTOR_LIST command) to the expected range of 0–1 for the PVECLIST GSR procedure before sending.

- If the value is less than 0 or greater than 720, clamp it to the nearest in-range value.
- If the value is greater than or equal to 360, subtract 360.
- Divide the value by 768.
- If the original value was greater than or equal to 360, add .5 to the result of the division.

This has the effect of mapping hue values in the range (0–360) to (0–.46875), and values in the range (360–720) to (.5–.96875). Values greater than .46875 and less than .5 are out of range, and are interpreted as .5 (pure blue).

If the vector class is "tabulated," the fourth position of the VECTORS is an INDEX. Users should specify whole numbers $0 \leq$ index ≤ 127 in this case. The GSRs will truncate the value supplied to an integer and force the value to be in range 0 to 127.

If specifying P_Conn, P_Dots, or P_Sepa, the vector's draw section of the vector list is generated by the procedure. P_Item and P_Tab require that the move/draw nature of each vector be defined by the user.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PVecList (      NumberOfVectors    : INTEGER;
                          VAR Vectors        : P_VectorListType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to send a piece of a vector list. For vector-normalized vector lists, this procedure can be called repeatedly to send the vector list down in pieces. Multiple calls to this procedure are not permitted for the block-normalized vector list case, unless the procedure PVecMax is called first. To send a vector list, the user must call the procedures:

PVecBegn

PVecList (This procedures may be called multiple times for vector-normalized vector lists)

PVecEnd

Together, the above 3 procedures implement the PS 300 command:

Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE, TABULATED) N=n <vectors>;

Vectors is the array containing the vectors of the vector list.

where: Vectors [n].V4[1] := Vector n x-component
Vectors [n].V4[2] := Vector n y-component
Vectors [n].V4[3] := Vector n z-component
Vectors [n].V4[4] := Vector n intensity (or hue)
0 <= vectors [n].V4[4] <=1 or 0<= Vectors[n].V4[4] <=127 if vector class is tabulated.

Vectors [n].Draw := True if vector n is a draw/line vector.
Vectors [n].Draw := False if vector n is a move/position vector.

The fourth position of Vectors is the intensity of that vector if vector-normalized, regardless of dimension. If block-normalized, the first vector's fourth position is used as the entire vector list intensity.

The fourth position of Vectors can be used to specify color in lieu of intensity when specifying color-blended vectors (refer to PSETBLND). Use the following algorithm to convert the acceptable range of hues (real numbers 0-720 for the PS 300 VECTOR_LIST command) to the expected range of 0-1 for the PVECLIST GSR procedure before sending.

- If the value is less than 0 or greater than 720, clamp it to the nearest in-range value.
- If the value is greater than or equal to 360, subtract 360.
- Divide the value by 768.
- If the original value was greater than or equal to 360, add .5 to the result of the division.

This has the effect of mapping hue values in the range (0-360) to (0-.46875), and values in the range (360-720) to (.5-.96875). Values greater than .46875 and less than .5 are out of range, and are interpreted as .5 (pure blue).

If the vector class is "tabulated," the fourth position of the VECTORS is an INDEX. Users should specify whole numbers $0 \leq$ index ≤ 127 in this case. The GSRs will truncate the value supplied to an integer and force the value to be in range 0 to 127.

If specifying P_Conn, P_Dots, or P_Sepa, the vector's draw section of the vector list is generated by the procedure. P_Item and P_Tab requires that the move/draw nature of each vector be defined by the user.

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
    [GLOBAL, CHECK(NOBOUNDS)] PROCEDURE PVecMax (Maxcomp : REAL)
             (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to set the maximum component of a vector list for multiple calls to PVecList with block-normalized vectors. To send a vector list, the user must call:

- PVecBegn
- PVecMax (If defining block-normalized vector with multiple calls to PVecList)
- PVecList (This may be called multiple times.)
- PVecEnd (This is called last.)

Together, the above 4 procedures implement the PS 300 command

```
Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE)
        N=n <vectors>;
```

Name := VECTOR_LIST (no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PVecMax (Maxcomp : REAL)
        (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure must be called to set the maximum component of a vector list for multiple calls to PVecList with block-normalized vectors. To send a vector list, the user must call:

- PVecBegn
- PVecMax (If defining block normalized-vector with multiple calls to PVecList)
- PVecList (This may be called multiple times.)
- PVecEnd (This is called last.)

Together, the above 4 procedures implement the PS 300 command

```
Name := VECTOR_LIST (DOTS, CONNECTED, ITEMIZED, SEPARATE)
        N=n <vectors>;
```

Name := WRITEBACK

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PWRTBACK (   CONST Name    : STRING;
                       CONST Name1   : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DESCRIPTION

This procedure enables writeback in the data structure **Name1**. Writeback is triggered by sending a TRUE to the writeback operation node created with this procedure.

PARAMETERS

Name1 – The name of the structure to which writeback is applied.

PS 300 COMMAND AND SYNTAX

```
name := WRITEBACK [APPLied to Name1];
```

Name := WRITEBACK

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PWrtBack (  %DESCR Name     : P_VaryingType;
                      %DESCR Name1    : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure enables writeback in the data structure Name1. Writeback is triggered by sending a TRUE to the writeback operation node created with this procedure.

PARAMETERS

Name1 - The name of the structure to which writeback is applied.

PS 300 COMMAND AND SYNTAX

name := WRITEBACK [APPLied to Name1];

APPLICATION SUBROUTINE AND PARAMETERS

CALL PWRTBK (Name, Name1, Errhnd)

where:

Name1 is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DEFINITION

This subroutine enables writeback in the data structure **Name1**. Writeback is triggered by sending a TRUE to the writeback operation node created with this subroutine.

PARAMETERS

Name1 - The name of the structure to which writeback is applied.

PS 300 COMMAND AND SYNTAX

name := WRITEBACK [APPLied to **Name1**];

Name := SET BLINKING ON/OFF

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSEBOF (Name, OnOff, Name1, Errhnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1
Name1 is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DEFINITION

This procedure turns blinking on and off. It affects all objects below the node created by the command in the display tree.

PARAMETERS

OnOff - TRUE indicates that blinking will occur in the displayed objects. FALSE turns blinking off.

Name1 - The name of the structure that will be affected by the command.

PS 300 COMMAND AND SYNTAX

```
name := SET BLINKing switch
        [APPLied to name1];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetBOnf  (  CONST Name      : STRING;
                             Onoff     : BOOLEAN;
                       CONST Name1     : STRING;
                     Procedure Error_Handler (Err : INTEGER ));
```

DEFINITION

This procedure turns blinking on and off. It affects all objects below the node created by the command in the display tree.

PARAMETERS

OnOff - TRUE indicates that blinking will occur in the displayed objects. FALSE turns blinking off.

Name1 - The name of the structure that will be affected by the command.

PS 300 COMMAND AND SYNTAX

name := SET BLINKing `switch` [APPLied to `name1`];

Name := SET BLINKING ON/OFF

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetBOnf (  %DESCR Name     : P_VaryingType;
                             Onoff    : BOOLEAN;
                      %DESCR Name1    : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure turns blinking on and off. It affects all objects below the node created by the command in the display tree.

PARAMETERS

Onoff – TRUE indicates that blinking will occur in the displayed objects. FALSE turns blinking off.

Name1 – The name of the structure that will be affected by the command.

PS 300 COMMAND AND SYNTAX

name := SET BLINKing **switch** [APPLied to **name1**];

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSELNT (Name, Pattrn, Cont, Namel, Errhnd)
```

where:

Name is a CHARACTER STRING
Pattrn is an INTEGER*4
Cont is a LOGICAL*1
Namel is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DEFINITION

This subroutine specifies the line texture pattern to be used in drawing the vector lists that appear below the node created by this command. There are up to 127 hardware-generated line textures possible. The parameter pattrn is an integer between 1 and 127. The desired line texture is indicated by the setting or clearing of the lower 7-bit positions in Pattern when represented in binary. An individual pattern unit is 1.1 centimeters in length. Some of the more common patterns and their corresponding bit settings are shown below:

Pattern	Bit representation	Line Texture repeated twice	
127	1111111	--------------	Solid
124	1111100	----- -----	Long Dashed
122	1111010	---- - ---- -	Long Short Dashed
106	1101010	-- - - -- - -	Long Short Short Dashed

PARAMETERS

Cont_ - LOGICAL value used to set a flag to indicate if the specified line texture should continue from one vector to the next. If Cont is TRUE, the line texture will continue from one vector to the next through the endpoint. If Cont is FALSE, the line texture will start and stop and the vector endpoints.

Pattrn - An integer between 1 and 127 that specifies the desired line texture. When pattern is less that 1 or greater than 127, solid lines are produced.

Namel - The name of the structure to which the line texture is applied.

DEFAULTS

The default line texture is a solid line

NOTES

Since 7 bit positions are used, it is not possible to create a symmetric pattern.

When line-texturing is applied to a vector, the vector that is specified is displayed as a patterned, rather that solid line. If the line is smaller than the pattern length, then as much of the pattern that can be displayed with the vector is displayed. If the line is smaller than the smallest element of the pattern, then the line is displayed as solid.

The **With Pattern** and **curve** commands create multiple vectors in memory. To the line-texturing hardware, each vector in a pattern or curve is seen as an individual vector. Line-texturing a patterned line or curve is the same as line-texturing a number of small segments. Curves and patterns affect line-texturing only in that they tend to create short vectors that may be too short to be completely textured.

PS 300 COMMAND AND SYNTAX

```
name := SET LINe_texture [AROUnd_corners] pattern
        [APPLied to name1];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetLinT  (  CONST Name       : STRING;
                          Pattern       : INTEGER;
                          AroundCorners : BOOLEAN;
                       CONST Name1      : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies the line texture pattern to be used in drawing the vector lists that appear below the node created by this command. There are up to 127 hardware-generated line textures possible. The parameter **pattern** is an integer between 1 and 127. The desired line texture is indicated by the setting or clearing of the lower 7 bit positions in **pattern** when represented in binary. An individual pattern unit is 1.1 centimeters in length. Some of the more common patterns and their corresponding bit settings are shown below:

Pattern	Bit representation	Line Texture repeated twice	
127	1111111	---------------	Solid
124	1111100	----- -----	Long Dashed
122	1111010	---- - ---- -	Long Short Dashed
106	1101010	-- - - -- - -	Long Short Short Dashed

PARAMETERS

AROUnd_corners – Boolean value used to set a flag to indicate if the specified line texture should continue from one vector to the next. If AROUnd_corners is TRUE, the line texture will continue from one vector to the next through the endpoint. If AROUnd_corners is FALSE, the line texture will start and stop at the vector endpoints.

Pattern – An integer between 1 and 127 that specifies the desired line texture. When **pattern** is less that 1 or greater than 127, solid lines are produced.

Name1 – The name of the structure to which the line texture is applied.

DEFAULTS

The default line texture is a solid line.

NOTES

Since 7 bit positions are used, it is not possible to create a symmetric pattern.

When line-texturing is applied to a vector, the vector that is specified is displayed as a textured, rather that solid line. If the line is smaller than the pattern length, then as much of the pattern that can be displayed with the vector is displayed. If the line is smaller than the smallest element of the pattern, then the line is displayed as solid.

The **With Pattern** and **curve** commands create multiple vectors in memory. To the line-texturing hardware, each vector in a pattern or curve is seen as an individual vector. Line-texturing a patterned line or curve is the same as line-texturing a number of small segments. Curves and patterns affect line-texturing only in that they tend to create short vectors that may be too short to be completely textured.

PS 300 COMMAND AND SYNTAX

```
name := SET LINe_texture [AROUnd_corners] pattern
                         [APPLied to name1];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSetLinT  (  %DESCR Name        : P_VaryingType;
                          Pattern        : INTEGER;
                          AroundCorners  : BOOLEAN;
                       %DESCR Name1       : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure specifies the line texture pattern to be used in drawing the vector lists that appear below the node created by this command. There are up to 127 hardware-generated line textures possible. The parameter **pattern** is an integer between 1 and 127. The desired line texture is indicated by the setting or clearing of the lower 7-bit positions in Pattern when represented in binary. An individual pattern unit is 1.1 centimeters in length. Some of the more common patterns and their corresponding bit settings are shown below:

Pattern	Bit representation	Line Texture repeated twice	
127	1111111	--------------	Solid
124	1111100	----- -----	Long Dashed
122	1111010	---- - ---- -	Long Short Dashed
106	1101010	-- - - -- - -	Long Short Short Dashed

PARAMETERS

AROUnd_corners – Boolean value used to set a flag to indicate if the specified line texture should continue from one vector to the next. If AROUnd_corners is TRUE, the line texture will continue from one vector to the next through the endpoint. If AROUnd_corners is FALSE, the line texture will start and stop at the vector endpoints.

Pattern – An integer between 1 and 127 that specifies the desired line texture. When **pattern** is less that 1 or greater than 127, solid lines are produced.

Name1 – The name of the structure to which the line texture is applied.

DEFAULTS

The default line texture is a solid line.

NOTES

Since 7 bit positions are used, it is not possible to create a symmetric pattern.

When line-texturing is applied to a vector, the vector that is specified is displayed as a textured, rather that solid line. If the line is smaller than the pattern length, then as much of the pattern that can be displayed with the vector is displayed. If the line is smaller than the smallest element of the pattern, then the line is displayed as solid.

The With Pattern and curve commands create multiple vectors in memory. To the line-texturing hardware, each vector in a pattern or curve is seen as an individual vector. Line-texturing a patterned line or curve is the same as line-texturing a number of small segments. Curves and patterns affect line-texturing only in that they tend to create short vectors that may be too short to be completely textured.

PS 300 COMMAND AND SYNTAX

```
name := SET LINe_texture [AROUnd_corners] pattern
                         [APPLied to name1];
```

Name := SET BLINK RATE

APPLICATION SUBROUTINE AND PARAMETERS

CALL PSEBR (Name, Rate, Name1, Errhnd)

where:

Name is a CHARACTER STRING
Rate is an INTEGER*4
Name1 is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine specifies the blinking rate in refresh cycles to be applied to all objects below the node created by the command in the display tree.

PARAMETERS

Rate – An integer designating the duration of the blink in refresh cycles. The blinking data will be on for the number of specified refreshes and off for the specified number of refreshes.

Name1 – The name of the structure to which the blinking rate is applied.

NOTE

PS 330 style blinking, done via the SET RATE and IF PHASE ON/OFF commands, where blinking is tied to the update rate rather than the refresh rate, will still work, but since the update rate in the PS 350 may be slower, the visual result may be different.

PS 300 COMMAND AND SYNTAX

```
name := SET BLINK RATE n
        [APPLied to name1];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSETBR (   CONST Name     : STRING;
                       Blinkrate    : INTEGER;
                     CONST Name1    : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DESCRIPTION

This procedure specifies the blinking rate in refresh cycles to be applied to all objects below the node created by the command in the display tree.

PARAMETERS

Blinkrate – An integer designating the duration of the blink in refresh cycles. The blinking data will be on for the specified number of refreshes and off for the specified number of refreshes.

Name1 – The name of the structure to which the blinking rate is applied.

NOTE

PS 330-style blinking, done via the SET RATE and IF PHASE ON/OFF commands, where blinking is tied to the update rate rather than the refresh rate, will still work, but since the update rate in the PS 350 may be slower, the visual result may be different.

PS 300 COMMAND AND SYNTAX

```
name := SET BLINK RATE n
        [APPLied to name1];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSETBR (   %DESCR Name      : P_VaryingType;
                        Blinkrate     : INTEGER;
                     %DESCR Name1     : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

DESCRIPTION

This procedure specifies the blinking rate in refresh cycles to be applied to all objects below the node created by the command in the display tree.

PARAMETERS

Blinkrate – An integer designating the duration of the blink in refresh cycles. The blinking data will be on for the specified number of refreshes and off for the specified number of refreshes.

name1 – The name of the structure to which the blinking rate is applied.

NOTE

PS 330-style blinking, done via the SET RATE and IF PHASE ON/OFF commands, where blinking is tied to the update rate rather than the refresh rate, will still work, but since the update rate in the PS 350 may be slower, the visual result may be different.

PS 300 COMMAND AND SYNTAX

```
name := SET BLINKING RATE n
        [APPLied to name1];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE   PViewL (  CONST Name    : STRING;
                            Hmin    : REAL;
                            Hmax    : REAL;
                            Vmin    : REAL;
                            Vmax    : REAL;
                            Imin    : REAL;
                            Imax    : INTEGER;
                      CONST Name1   : STRING;
                    Procedure Error_Handler (Err : INTEGER));;
```

DEFINITION

The PViewL procedure for the PS 350 loads a viewport and overrides the concatenation of the previous viewport. As with the standard PS 300 VIEWPORT command, it specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines. It affects all objects below the node created by the command in the display tree.

PARAMETERS

Hmin,Hmax,Vmin,Vmax – The x and y boundaries of the new viewport. Values must be within the -1 to 1 range.

Imin,Imax – Specifies the minimum and maximum intensities for the viewport. **imin** is the intensity of lines at the back clipping plane; **imax** at the front clipping plane. Values must be within the 0 to 1 range.

Name1 – The name of the structure to which the viewport is applied.

PS 300 COMMAND AND SYNTAX

```
name := LOAD VIEWport HORizontal = hmin:hmax
        VERTical = vmin:vmax
        [INTENsity = imin:imax] [APPLied to name1];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE   PViewL  ( %DESCR Name      : P_VaryingType;
                             Hmin      : REAL;
                             Hmax      : REAL;
                             Vmin      : REAL;
                             Vmax      : REAL;
                             Imin      : REAL;
                             Imax      : INTEGER;
                      %DESCR Name1     : P_VaryingType;
                     Procedure Error_Handler (Err : INTEGER ));;
```

DEFINITION

The PViewL procedure for the PS 350 loads a viewport and overrides the concatenation of the previous viewport. As with the standand PS 300 VIEWPORT command, it specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines.

PARAMETERS

Hmin,Hmax,Vmin,Vmax - The x and y boundaries of the new viewport. Values must be within the -1 to 1 range.

Imin,Imax - Specifies the minimum and maximum intensities for the viewport. imin is the intensity of lines at the back clipping plane; imax at the front clipping plane. Values must be within the 0 to 1 range.

Name1 - The name of the structure to which the viewport is applied.

PS 300 COMMAND AND SYNTAX

```
name := LOAD VIEWport HORizontal = hmin:hmax
        VERTical = vmin:vmax
        [INTENsity = imin:imax] [APPLied to name1];
```

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PVIEWL  ( Name, Hmin, Hmax, Vmin, Vmax, Imin, Imax, Namel, Errhnd)
```

where:

Hmin, Hmax are REAL*4
Vmin, Vmax are REAL*4
Imin, Imax are REAL*4
Namel is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

DEFINITION

The PViewL subroutine for the PS 350 loads a viewport and overrides the concatenation of the previous viewport. As with the standand PS 300 VIEWPORT command, it specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines.

PARAMETERS

Hmin,Hmax,Vmin,Vmax – The x and y boundaries of the new viewport. Values must be within the -1 to 1 range.

Imin,Imax – Specifies the minimum and maximum intensities for the viewport. **imin** is the intensity of lines at the back clipping plane; **imax** at the front clipping plane. Values must be within the 0 to 1.

Namel – The name of the structure to which the viewport is applied.

PS 300 COMMAND AND SYNTAX

```
name := LOAD VIEWport HORizontal = hmin:hmax
        VERTical = vmin:vmax
        [INTENsity = imin:imax] [APPLied to namel];
```

PS 300 WRITEBACK FEATURE

The Writeback feature allows displayed transformed vector data to be sent back to the host. The position of the writeback node in the display structure determines which transformations will be applied to the writeback data. The system-generated writeback node will include all transformations (viewing and modeling). Once the host has received these data, they can be used to generate hardcopy plots or display host-generated raster images. The user is responsible for retrieval and all subsequent processing of data on the host system.

This guide describes how to use the Writeback feature on all members of the PS 300 family of graphics computers. Operational differences among models are specifically noted.

This guide contains:

- A description of the user interface for the Writeback feature. The user interface consists of the WRITEBACK operation node and the WRITEBACK initial function.
- Constraints on the use of the WRITEBACK operation node.
- Descriptions of the WRITEBACK function.
- A list of the commands that may need to be interpreted by host-resident code to filter writeback data retrieved from the PS 300.
- An example of the sequence of data sent back to the host.
- An example of a host program that retrieves, processes, and files writeback data from the PS 350.

Change-pages supporting the Writeback feature are provided in this guide for the Command Summary, the Function Summary and the Graphics Support Routine sections of the PS 300 Document Set.

Writeback User Interface

The Writeback feature is implemented by:

- Creating the WRITEBACK operation node (or using the system-generated writeback node, WB$).
- Activating the WRITEBACK operation node.
- Connecting the WRITEBACK function to a function network.

WRITEBACK Operation Node

When the PS 300 is booted, a WRITEBACK operation node is created. It is named WB$ and is placed above every user-defined display structure. This node can be triggered if an entire displayed picture is to be included in the writeback data. If writeback of only a portion of the picture is desired, the user must place other WRITEBACK nodes appropriately in the display structure.

A user-defined WRITEBACK operation node is created by the command:

```
Name := WRITEBACK [APPlied to Name1];
```

The WRITEBACK node has one input. A TRUE sent to input <1> of the WRITEBACK node triggers writeback for the data structure below the node. This trigger is sent by the user, for example:

```
SEND TRUE TO <1>name;
```

triggers that WRITEBACK node. Of course the node could be triggered through a function network using a function key, etc.

A WRITEBACK operation node delimits the structure from which the writeback data will be collected. Only the data nodes below the WRITEBACK operation node in the display structure will be transformed, clipped, viewport scaled perspective divided (as delineated by the placement of the WRITEBACK node), and sent back to the host.

NOTE

On the PS 350, viewport translations will not be applied to the data.

WRITEBACK Operation Node Constraints

Only a displayed structure can be enabled for writeback. This means that the WRITEBACK operation node must be traversed by the display processor and therefore must be included in the displayed portion of the structure. The default WRITEBACK node WB$ is displayed as part of every displayed structure. But, if the user creates another WRITEBACK node and if this node is triggered before being displayed, the following error message will result:

E 8 ACP cannot find your operate node

Any number of WRITEBACK nodes can be placed within a structure. However, only one WRITEBACK operation can occur at a time. If more than one node is triggered, the WRITEBACK operations are performed in the order in which the corresponding nodes were triggered.

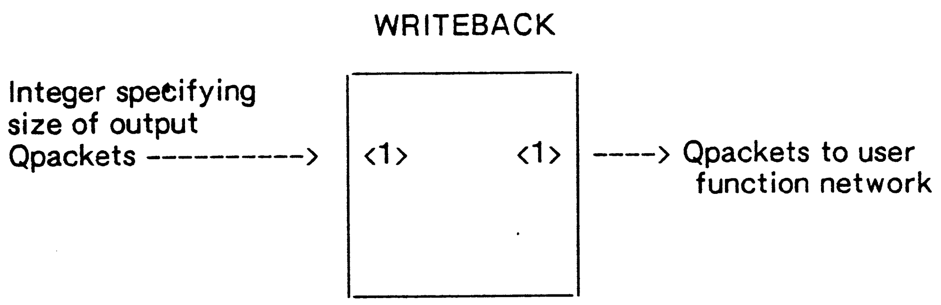
The terminal emulator and message_display information will not be returned to the host.

Polygon data can be returned to the host only if the PS 340 has a 4K ACP.

Before triggering the WRITEBACK operation, disable the SCREENSAVE function by entering the command "SCREENSAVE:= nil;".

The WRITEBACK Function

An initial function instance, WRITEBACK, is created by the system at boot up.


WRITEBACK
Integer specifying size of output Qpackets ---------->
<1>
<1>
----> Qpackets to user function network

WRITEBACK sends encoded writeback data received from the display processor. The writeback data is prefixed by a start-of-writeback command, followed by the encoded data, followed by an end-of-writeback or end-of-frame command.

WRITEBACK has one user-accessible input queue. Input <1> accepts integers specifying the size of Qpackets to be output by the function. The default size is 512 bytes per Qpacket. The minimum and maximum size are 16 bytes per Qpacket and 1024 bytes per Qpacket, respectively. If the size specified by the user is not within this range, the default size will be used by the system.

The input value should be chosen such that the actual size of the qpacket sent to the I/O port is less than or equal to the present input buffer size on the host computer.

If the CVT8TO6 function is used to send the binary data to the host, then the number of the bytes sent to the host is approximately 3/2 * the number of bytes sent by the Writeback function.

For example, if the integer sent to <1> of the Writeback function is 80, the largest Qpacket sent to the host will be 80 * 3/2 = 120. Qpackets, where the size is not a multiple of 4, will be padded to the next multiple of 4. For instance, Qpacket sizes of 77, 78, and 79, sent to CVT8TO6 will all have output sizes of 120.

WRITEBACK has one user-accessible output queue. Output <1> passes the encoded writeback data out as Qpackets until the end-of-writeback or end-of-frame command is seen.

This function is not activated by the normal input queue triggering mechanism. It is activated by sending a TRUE to any WRITEBACK operation node.

Data Output by WRITEBACK

WRITEBACK will return all data below the WRITEBACK operation node. Host-resident code will be responsible for recognizing the start-of-writeback and end-of-writeback or end-of-frame commands.

Attribute information, such as color, must be interpreted by host code to ensure that the hardcopy plots are correct.

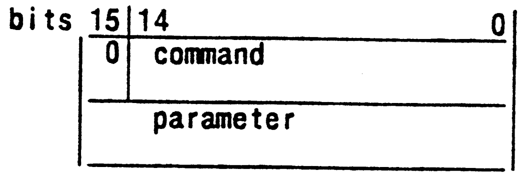
On the PS 350, viewport translations will not be applied to the data. Correct computation of the position of endpoints requires that the host program add a viewport center to each endpoint. The initial viewport center is established with a VIEWPORT CENTER command. The VIEWPORT CENTER command is sent following the start-of-writeback command. Any changes to the viewport center will be indicated through this sequence of commands: CLEAR DDA, CLEAR SAVE POINT, position endpoint, CLEAR SAVE POINT. The position endpoint becomes the new viewport center.

Also, on the PS 350, several commands such as ENABLE PICK and ENABLE BLINK are sent to the host. These will not typically be needed by the host program. However, these commands come directly from the refresh buffer and are not filtered by the PS 350. Host-resident code must filter the writeback data and strip out nonessential information.

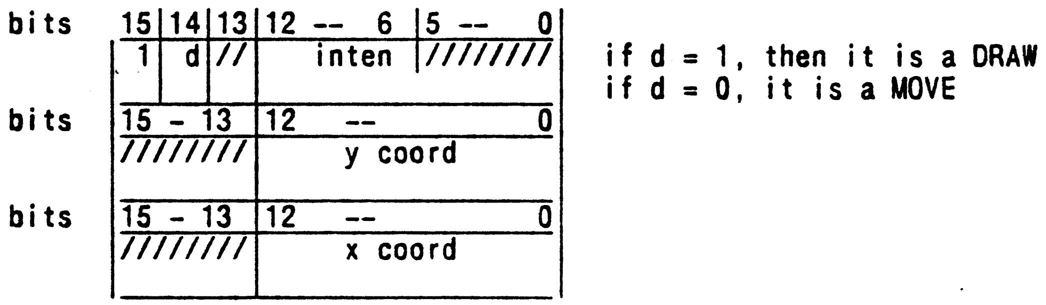
Data Packets Returned

Data packets sent out the WRITEBACK function contain the following information:

- If bit 15 of the first word is 0, it signals that the data that follows is a command. For example, if the first word is H#0200 (Hex 0200) then the Line Generator status will follow.



- If bit 15 of the first word is 1, it indicates that intensity, x and y coordinate information will follow. Intensity can range from 0 to 127. The format of the data is:



NOTE

In the illustrations of data format, the slash character is used to illustrate blocks of data that are unused.

Command Descriptions

The following list describes the commands that the host-resident code might have to interpret before it can recognize and filter writeback data received from the PS 300. These commands can be intermixed with vector data.

It is important to note that each command contains at least three 16-bit words. For example, if a command only has one parameter then the third word is unused, but it is still sent to the host. If a command has 3, 4, or 5 parameters, then 6 words will be sent for that command.

START-OF-WRITEBACK   code in hex = H#0B00
# 2816

Parameters:
Line texture (one word)
LGS (one word)

Marks the beginning of the writeback segment, of which there is guaranteed to be only one.

The texture and line generator status are included here. They follow the same format as the texture and line generator status shown below.

```
|        B00          |
|/////////| Texture   |
|        LGS          |
```

END-OF-WRITEBACK   code in hex = H#0C00
# 3072

Parameters:
None

Marks the end of the writeback segment. For the PS 350, the end-of-writeback may also be indicated by the end-of-frame command.

```
|         C00          |
|    0     |     0/1   |    0 = finished successfully, 1 = cannot finish
|//////////////////////|        operation because of insufficient memory
```

The error code (0 or 1) is currently not present in the PS 350 systems.

LINE GENERATOR STATUS   code in hex = H#0200
# 512

Parameters:
Status word (one word)

Indicates dot mode (bit 8) and which display is selected (bits 0-3). Normally, only the dot mode bit must be referenced.

```
|         200          |
|         LGS          |
|//////////////////////|
```

Line Generator Status Register (LGS):

```
|///|///|///|///|///|///|///|SHO|///|///|///////|   SCOPE SELECT  |
|///|///|///|///|///|///|///|EPT|///|///|///////| D   C   B   A   |
 15  14  13  12  11  10  09  08  07  06  05  04  03  02  01  00
```

Bit	Logical Names	
		B A
08	SHOWENDPT	Dot mode
03	BLANKD	Blank scope D (1 blanks the scope 0 enables the scope)
02	BLANKC	Blank scope C
01	BLANKB	Blank scope B
00	BLANKA	Blank scope A

COLOR

code in hex = H#0400
# 1024

Parameters:
Color value (one word)

```
|        400          |
|   Hue   | Saturation|
|/////////////////////|
```

```
|///                             |///             ///////////|
|/// HI        HUE           LO  |//// HI  SAT    LO ///////////|
 15  14  13  12  11  10  09  08  07  06  05  04  03  02  01  00
```

TEXTURE

code in hex = H#0500
# 1280

Parameters:
Texture value (one word)

```
|        500          |
|/////////| Texture   |
| ////////////////////|
```

Line Generator Texture Register:

```
|///////////////////////////////////|   Texture bit pattern        |
|///////////////////////////////////|                              |
 15  14  13  12  11  10  09  08  07  06  05  04  03  02  01  00
```

H#007F or H#00FF both default to a Solid line.
For non-PS 350 users, the texture will always be H#00FF.

The following commands are for PS 350 users ONLY.

CLEAR DDA — code in hex = H#0100
# 256

Parameters:
None

PICK BOUNDARY — code in hex = H#0300
# 768

Parameters:
Four Boundary Values (4 words)

CLEAR SAVE POINT — code in hex = H#0600
# 1536

Parameters:
None

SET PICK ID — code in hex = H#0700
# 1792

Parameters:
Pick ID Pointer (two words)

SET LightPen MODE — code in hex = H#0800
# 2048

Parameters:
Control Mask (1)
New X,Y (2)
Delta distance (1)
Delta frames (1) (Total five words)

ENABLE PICK — code in hex = H#0900
# 2304

Parameters:
None

DISABLE PICK — code in hex = H#0A00
# 2560

Parameters:
None

SET BLINK RATE — code in hex = H#0D00
# 3328

Parameters:
Blink Rate (one word)

ENABLE BLINK — code in hex = H#0E00
# 3584

Parameters:
None

DISABLE BLINK — code in hex = H#0F00
# 3840

Parameters:
None

END-OF-FRAME — code in hex = H#1700
# 5888

Parameters:
None

Signifies that the current update cycle is completed and that any following data is part of the next update frame. This also signifies end of the writeback segment.

VIEWPORT CENTER — code in hex = H#1800

Parameters:
x center (one word)
y center (one word)
z center (one word)
spare (two words)

```
  bits 15 ..................... 0
       | coordinates            |  2's complement vector
       |________________________|
```

This value has to be added to each x,y coordinate pair. This information is necessary to calculate the actual coordinates of the data which has been viewport scaled. Every time a new viewport is traversed by the Arithmetic Control Processor, a new viewport center command will be sent.

NOTE

Codes H#1900 - H#1F00 are reserved for future commands. Code H#0000 is defined as a no-op, and naturally has no parameters.

EXAMPLE OF THE SEQUENCE OF DATA SENT BACK TO THE HOST

The following example illustrates the sequence of data and the data in byte format sent to the host during a WRITEBACK operation.

B00	Start-of-writeback command
////////// Texture	
LGS	
400	Color command
Hue Saturation	
/////////////////////////	
Intensity	V
Y	E
X	C
:	T
:	O
:	R
:	S
:	
:	
:	
:	
200	Line Generator Status command
LGS	
/////////////////////////	
500	Texture command
////////// Texture	
/////////////////////////	
400	Color command
Hue Saturation	
/////////////////////////	
Intensity	V
Y	E
X	C
:	T
:	O
:	R
:	S
:	
:	
:	
:	
C00	End-of-writeback command
0/1	0 = finished successfully, 1 = cannot
/////////////////////////	finish because of insufficient memory

Data in Byte Format

```
0B  00    Start-of-writeback command
00  FF    Texture
04  70    LGS
04  00    Color command
80  00    Hue/Saturation
00  00    Not used
00  FF    Intensity
1Y  FF       Y
1X  FF       X
00  FF    Intensity
2Y  FF       Y
2X  FF       X
 :           :
 :           :
 :           :
02  00    LGS command
04  70    LGS
00  00    Not used
05  00    Texture command
00  FF    Texture
00  00    Not used
04  00    Color command
80  00    Color
00  00    Not used
00  FF    Intensity
1Y  FF       Y
1X  FF       X
 :           :
 :           :
 :           :
0C  00    End-of-writeback command
00  00    Finshed successfully
00  00    Not used.
```

SAMPLE WRITEBACK PROGRAM

```
PROGRAM Writeback(Input,Output,Outfile,Devfile);
{ Program to read writeback data from a PS 350. This program sets up a  }
{ function network to get the writeback data and processes the data and }
{ creates a data file on the host with the data from the PS 350.        }

CONST
 %INCLUDE 'PROCONST.PAS'
 Max_buf = 1024;

TYPE
 Int16 = -32768..32767;
 Max_line = VARYING [Max_buf] OF CHAR;
 %INCLUDE 'PROTYPES.PAS'

VAR
 OUTFILE : TEXT;
 DEVFILE : TEXT;
 DEVSPEC : P_VARYINGTYPE;
 OUTNAME : P_VARYINGTYPE;
 WBNAME  : P_VARYINGTYPE;
 COMMAND : INT16;
 INDEX : INTEGER;
 LEN : INTEGER;
 Inline : P_VARYBUFTYPE;
 vx,vy,vz : REAL;
 In_DDA : BOOLEAN := FALSE;

 %INCLUDE 'PROEXTRN.PAS'

  PROCEDURE ERR (ERROR: INTEGER);
  {}
  { ERROR HANDLER ROUTINE }
  {}
    BEGIN { ERR }
      {}
      WRITELN(' ERROR :=',ERROR);
      HALT;
      {}
    END; { ERR }
  PROCEDURE Setup;
  { Create function network to send writeback data to host }
  { This uses F:cvt8to6 to send 6-bit data to the host }
      BEGIN
      PFnInst('cvt','cvt8',Err);
      Pconnect ('Writeback',1,1,'cvt',Err);
      Pconnect ('cvt',1,1,'host_message', Err);
      PsndStr (CHR(36),2,'cvt',Err);
      PsndFix (48,1,'writeback', Err);
      PNameNil('screensave',Err);
      PPurge( Err);
      END;

{ Utility procedures}
  PROCEDURE Six_to_eight( Inbuf :  Max_line;
   VAR Outbuf : P_VARYBUFTYPE);
  { Data from PS 350 is in six-bit packed format. This procedure unpacks
    data}

  CONST Base = 36;

  TYPE
    Cheat_4 = PACKED RECORD CASE Boolean OF
   TRUE : ( i: UNSIGNED);
   FALSE : ( c: PACKED ARRAY [1..4] OF CHAR);
 END;

  VAR
    w : Cheat_4;
    c_out,cycle_count,buf_index,il,tc : INTEGER;
    first : BOOLEAN;

  BEGIN
    buf_index := 1;
    first := TRUE;
    cycle_count := 1;
    c_out := 4;
    outbuf := '';
    WHILE buf_index <= len DO
      BEGIN
 tc := ORD(Inbuf[buf_index]) - base;
 IF first THEN
   IF tc < 0 THEN
     c_out := 4+tc
   ELSE
     BEGIN
       first := FALSE;
       w.i := tc;
       cycle_count := SUCC(cycle_count);
     END { ELSE tc >= 0 }
```

```
ELSE
  BEGIN
    w.i := w.i * (2**6);
    w.i := UOR(w.i ,tc);
    cycle_count := SUCC(cycle_count);
  END; { ELSE }
IF cycle_count > 6 THEN
  BEGIN
    FOR il := 4 DOWNTO (5-c_out) DO
      Outbuf := outbuf + w.c[il];
    cycle_count := 1;
    first := true;
  END;
buf_index := SUCC(buf_index);
     END; { WHILE }
 END;

 PROCEDURE Next_Block;
 { Get a block of data from the PS 350 and convert from six to eight}
 { bit format }

 VAR Inbuff : Max_line;

 BEGIN
   PGETWAIT(Inbuff,err);
   Index := 1;
   Len := LENGTH(Inbuff);
   Six_to_eight ( Inbuff, Inline);
   Len := LENGTH(Inline);
 END;

 PROCEDURE Get_Value( VAR a : INT16);
 { Convert two bytes of input buffer to 16 bit integer }

 VAR i : INTEGER;

 BEGIN { Get_Value }
   a := 0;
   FOR i := 1 TO 2 DO
     BEGIN
Index := Index + 1;
IF Index > Len THEN
  Next_Block;
a := a * 256 + ORD(Inline[Index]);
     END;
 END;{ Get_Value }
```

```
{ Procedures for processing refresh buffer commands }

  PROCEDURE Clear_DDA;
  { CLEAR DDA - %X0100 }
  { Parameters - None }
  { Indicates start of sequence to set viewport center }
  { This sequence is CLEAR DDA, CLEAR SAVE POINT, Vector, CLEAR SAVE POINT}

  VAR a,b : Int16;

  BEGIN
    In_DDA := TRUE;
    Get_value ( a );
    Get_value ( b );
    Writeln(Outfile,'{Clear DDA}');
  END;

  PROCEDURE Write_LGS;
  { WRITE LINE GENERATOR STATUS - %X0200 }
  { Parameters - Status word (one word)  }
  { Bit   8 : Dot mode.  }
  { Bit   6 : Fast sweep ( Opposite of 7) }
  { Bits  5 -  4: Contrast selection (00-min,11-max)}
  { Bits  3 -  0: Scope select( 1 disables,0 enables)}

  VAR lgs,a : Int16;

  BEGIN
    Get_value ( lgs );
    Get_value ( a );
    Writeln(Outfile,'{Write LGS:',HEX(lgs),'}');
  END;

  PROCEDURE Write_Pick_Bound;
  { WRITE PICK BOUNDARY - %X0300  }
  { Parameters - Left, Right, Bottom, Top }

  VAR l,r,b,t,a : Int16;

  BEGIN
    Get_value ( l );
    Get_value ( r );
    Get_value ( b );
    Get_value ( t );
    Get_value ( a );
    Writeln(Outfile,'{Write_Pick_bound:',HEX(l),HEX(r),HEX(b),HEX(t),'}');
  END;
```

```
PROCEDURE Write_Color;
{ WRITE COLOR - %X0400  }
{ Parameters - Color value (one Word) }
{ Bit  15 : Not Used  }
{ Bits 14 - 8 : Hue (High order in 14)}
{ Bit   7 : Not Used  }
{ Bits  6 - 3 : Sat (High order in 3) }
{ Bits  2 - 0 : Not Used  }

VAR c,a : Int16;

BEGIN
  Get_value ( c );
  Get_value ( a );
  Writeln(Outfile,'{Write_Color:',HEX(c),'}');
END;

PROCEDURE Write_Texture;
{ WRITE TEXTURE - %X0500      }
{ Parameters - Texture value (one word)   }
{ Bits 15 - 7 : Not Used      }
{ Bits  6 - 0 : Texture bit pattern     }

VAR t,a : Int16;

BEGIN
  Get_value ( t );
  Get_value ( a );
  Writeln(Outfile,'{Write_Texture:',HEX(t),'}');
END;

PROCEDURE Clear_Save_Point;
{ CLEAR SAVE POINT - %X0600 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Clear_Save_Point:}');
END;

PROCEDURE Set_Pick_Id;
{ SET PICK ID - %X0700      }
{ Parameters - Pick Id Pointer (two words)}

VAR a,b : Int16;
```

```
BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Set_Pick_Id:',HEX(a),HEX(b),'}');
END;

PROCEDURE Set_Lightpen_Mode;
{ SET LIGHTPEN MODE - %X0800 }
{ Parameters - Control mask     }
{   Tracking cross y    }
{   Tracking cross x    }
{   Delta distance      }
{   Delta frames     }

VAR cm,x,y,dd,df : Int16;

BEGIN
  Get_value ( cm );
  Get_value ( x );
  Get_value ( y );
  Get_value ( dd );
  Get_value ( df );
  Writeln(Outfile,'{Set_Lightpen_mode:',HEX(cm),HEX(x),HEX(y),
    HEX(dd),HEX(df),'}');
END;

PROCEDURE Enable_Pick;
{ ENABLE PICK - %X0900}
{ Parameters - None }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Enable_Pick:}');
END;

PROCEDURE Disable_Pick;
{ DISABLE PICK - %X0A00   }
{ Parameters - None     }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Disable_Pick:}');
END;
```

```
PROCEDURE Enable_Writeback;
{ ENABLE WRITEBACK - %X0B00 }
{ Parameters - Line Texture }
{   Line Gen Status}

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Enable_Writeback:',HEX(a),HEX(b),'}');
END;

PROCEDURE Disable_Writeback;
{ DISABLE WRITEBACK - %X0C00 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Disable_Writeback:}');
END;

PROCEDURE Set_Blink_Rate;
{ SET BLINK RATE - %X0D00 }
{ Parameters - Blink rate }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Set_Blink_Rate:',HEX(a),'}');
END;

PROCEDURE Enable_Blink;
{ ENABLE BLINK - %X0E00 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Enable_Blink:}');
END;
```

```
PROCEDURE Disable_Blink;
{ DISABLE BLINK - %X0F00 }
{ Parameters - None  }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{Disable_Blink:}');
END;

PROCEDURE End_Of_Frame;
{ END OF FRAME - %X1700    }
{ Parameters - None      }

VAR a,b : Int16;

BEGIN
  Get_value ( a );
  Get_value ( b );
  Writeln(Outfile,'{End_Of_Frame:}');
END;

PROCEDURE Viewport_Center;
{ VIEWPORT CENTER - %X1800}
{ Parameters - x center    }
{   y center    }
{   z center    }

VAR xc,yc,zc,a,b : Int16;

BEGIN
  Get_value ( xc );
  Get_value ( yc );
  Get_value ( zc );
  Get_value ( a );
  Get_value ( b );
  vx := xc;
  IF (vx >= 32768) THEN vx := vx - 65536.0;
  vx := vx/32767;
  vy := yc;
  IF (vy >= 32768) THEN vy := vy - 65536.0;
  vy := vy/32767;
  vz := zc;
  IF (vz >= 32768) THEN vz := vz - 65536.0;
  vz := vz/32767;
  Writeln(Outfile,'{Viewport_Center:',vx:6:6,' ',vy:6:6,' ',vz:6:6,'}');
END;
```

```
PROCEDURE Process_Vector;
{ Vector - Bit 15 of command = 1 }
{ Word 1 ( command )   }
{ Bit  15 : Always one for vector }
{ Bit  14 : 1 = Draw, 0 = Move }
{ Bits 12 - 6 : Intensity/2  }
{ Bits  5 - 0 : Not Used  }
{ Word 2 ( y coord)   }
{ Bits 15 - 13: Not Used  }
{ Bits 12 -  0: Y coordinate  }
{ Word 3 ( x coord)   }
{ Bits 15 - 13: Not Used  }
{ Bits 12 -  0: X coordinate  }

VAR a,b : Int16;
  un : UNSIGNED;
  pl : CHAR;
  int,x,y : REAL;

BEGIN
  Get_value ( a );
  Get_value ( b );
  un:=command;
  pl:='l';
  IF (UAND(un,%X4000) = 0) THEN pl := 'p';
  un := UAND(un,%X1FC0);
  int := un;
  IF In_DDA THEN
    vz := int/8128.0
  ELSE
    int := (int/8128.0 + vz) * 2;
  un := a;
  un := UAND(un,%X1FFF);
  y := un;
  IF (y >= %X1000) THEN y := y - %X2000;
  IF In_DDA THEN
    vy := y / %XFFF
  ELSE
    y := y / %XFFF + vy;
  un := b;
  un := UAND(un,%X1FFF);
  x := un;
  IF (x >= %X1000) THEN x := x - %X2000;
  IF In_DDA THEN
    vx := x / %XFFF
  ELSE
    x := x / %XFFF + vx;
  IF In_DDA THEN
    BEGIN
  Writeln(Outfile,'{New View Center:',vx:6:6,' ',vy:6:6,' ',vz:6:6,'}');
  In_DDA := FALSE;
      END
    ELSE
    Writeln(Outfile,'{Vec ',pl,' ',x,',',y,' i=',int,'}');
  END;

  PROCEDURE Unknown;
  VAR a,b : Int16;

  BEGIN
    Get_value ( a );
    Get_value ( b );
    Writeln(Outfile,'{Unknown:',HEX(command),HEX(a),HEX(b),'}');
  END;

BEGIN  { Writeback}
  Write ('Enter Output File Name:');
  Readln(Outname);
  Write ('Enter Writeback Operate Node Name:{WB$ is default mode}');
  Readln(wbname);
  open(Outfile,Outname,new);
  rewrite(Outfile);

  { Look for file specifying line for pattach procedure }
  { Example of record in PSDEV.DAT: }
  { 'logdevnam=tt:/Phydevtyp=async' }
  open(devfile,'psdev',old);
  reset(devfile);
  readln(devfile,devspec);
  close(devfile);

  PATTACH(devspec,err);  { Attach to PS 350 }
  Setup;   { Setup writeback network }

  PNAMENIL('SCREENSAVE', ERR);
  PPURGE(ERR);
  PSndBool(TRUE,1,wbname, Err); { Trigger write back operate }

  Next_block;   { Read in first block of writeback data}

  Index := 0;
  Command := 0;
  vx := 0.0;
  vy := 0.0;
  vz := 0.0;

  { Process writeback buffers until END OF FRAME or END WRITEBACK}
  WHILE (Command <> %X0C00) AND (Command <> %X1700) DO
```

```
    BEGIN
      Get_value(Command);
      IF (Command > 32767) THEN { If bit 15 of command if set}
 Process_vector
   ELSE
   CASE (Command DIV 256) OF
     %X01 : Clear_DDA;
     %X02 : Write_LGS;
     %X03 : Write_Pick_Bound;
     %X04 : Write_Color;
     %X05 : Write_Texture;
     %X06 : Clear_Save_Point;
     %X07 : Set_Pick_Id;
     %X08 : Set_Lightpen_Mode;
     %X09 : Enable_Pick;
     %X0A : Disable_Pick;
     %X0B : Enable_Writeback;
     %X0C : Disable_Writeback;
     %X0D : Set_Blink_Rate;
     %X0E : Enable_Blink;
     %X0F : Disable_Blink;
     %X17 : End_Of_Frame;
     %X18 : Viewport_Center;
     OTHERWISE Unknown;
   END; { CASE }
    END;
  PFNINST('SCREENSAVE', 'SCREENSAVE', ERR  PDETACH(ERR);
  PPURGE(ERR):
  {}
END.  { Writeback}
```

E&S CUSTOMER SERVICE TELEPHONE INFORMATION LIST

Evans & Sutherland Customer Engineering provides a central service numbered staffed by CE representatives who are available to take requests from 9:00 a.m. Eastern Time to 5:00 p.m. Pacific Time (7:00 a.m. to 6:00 p.m. Mountain Time). All calls concerning customer service should be made to one of the following numbers during these hours. Before you call, please have available your customer site number and system tag number. These numbers are on the label attached to your PS 300 display or control unit.

Customers in the continental United States should call toll-free:

1 + 800 + 582-4375

Customers within Utah or outside the continental United States should call Dispatch at:

(801) 582-9412

If problems arise during product installation or you have a question that has not been answered adequately by the customer engineer or the customer service center, contact the regional manager at one of the following Customer Engineering offices:

Eastern Regional Manager
(for Eastern and Central Time Zones)
(518) 885-4639

Western Regional Manager
(for Mountain and Pacific Time Zones)
(916) 448-0355

If the regional office is unable to resolve the problem, you may want to call the appropriate department manager at corporate headquarters:

National Field Operations
(for field service issues)
(801) 582-5847, ext 4843

Software Support
(for sofware issues)
(801) 582-5847, ext 4810

Technical Support
(for hardware issues)
(801) 582-5847, ext 4868

Director of Customer Engineering
(for any unresolved problem)
(801) 582-5847, ext 4840

READER COMMENT FORM **Publication Number** ____________

Title ____________

Your comments will help us provide you with more accurate, complete, and useful documentation. After making your comments in the space below, cut and fold this form as indicated, and tape to secure (please do not staple). This form may be mailed free within the United States. Thank you for your help.

How did you use this publication?

☐ General information	☐ As a reference manual
☐ Guide to operating instructions	☐ Other ____________

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR
Technical Accuracy Is the manual technically accurate?	☐	☐	☐	☐
Completeness Does the manual contain enough information?	☐	☐	☐	☐
Readability Is the manual easy to read and understand?	☐	☐	☐	☐
Clarity Are the instructions easy to follow?	☐	☐	☐	☐
Organization Is it easy to find needed information?	☐	☐	☐	☐
Illustrations and Examples Are they clear and useful?	☐	☐	☐	☐
Physical Attractiveness What do you think of the overall appearance?	☐	☐	☐	☐

What errors did you find in the manual? (Please include page numbers) ____________

Name ____________	**Street** ____________
Title ____________	**City** ____________
Department ____________	**State** ____________
Company ____________	**Zip Code** ____________

All comments and suggestions become the property of Evans & Sutherland.

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 4632 SALT LAKE CITY, UTAH

POSTAGE WILL BE PAID BY ADDRESSEE

EVANS & SUTHERLAND
580 Arapeen Drive
Salt Lake City, Utah 84108

ATTN: IAS TECHNICAL PUBLICATIONS

Fold

Cut along dotted line

Section RM16
Index

Indicators

Entries are indexed by volume, section, and page number. In cases where a topic appears on more than one successive page and discussion of it is continuous, page indicators only refer to the page of that discussion on which the topic first appears. There are no inclusive references. Information on a topic may be found on several successive pages following the page that is referenced. A reference to the first page of a section may indicate that the topic is discussed throughout that section.

A sample entry is:

Viewing operations, GT2–44; GT8–1
  attributes, GT8–48, 56
  commands, IS2–17
  default values, GT8–2, 52
  node, GT8–53, 54, 55

The first reference after the main entry is to the *Graphics Tutorial* volume, Section 2, page 44, where a discussion of viewing operations begins. The second reference is to the first page of Section 8 in the *Graphics Tutorial* volume. That entire section discusses viewing operations. The subentries refer to specific aspects of viewing operations. Note that in the case of the subentry "node," successive pages are indexed because the discussion of the topic is not continuous.

Alphabetization

Index entries, including abbreviations and acronyms, are alphabetized on a letter-by-letter basis. In the order of entries and subentries, numbers come before letters. There is one exception, in which a number begins a main entry. 03$ is alphabetized under Z, as if spelled out. Words are alphabetized up to the first mark of punctuation. Spaces between words, hyphens, slant lines, and underscores are ignored in the entry sequence.

A sample ordering of entries and subentries is:

```
Matrix
  2x2
  3x3
  4x4
  accumulated
  algebra
MATRIX2, F:
MATRIX_2x2
MATRIX3, F:
MATRIX_3x3
```

Cross-References

A *See* cross-reference is to the entry that has been chosen in cases where alternatives, such as synonyms or word order variants, existed. A *See also* cross-reference refers to an entry where indicators to additional or related information can be found. Not all related topics are cross-referenced. Note especially that topics whose entries appear in close proximity in the index (such as "Viewing operations" and "Viewing area") are not cross-referenced. In most cases, there are no cross-references from a subentry to a main entry with the same wording. For example, there is no cross-reference from the main entry "Display structure," subentry "conditional referencing" to the main entry "Conditional referencing."