PS 390 DOCUMENT SET

REFERENCE MATERIALS 5–16

Copyright © 1987
EVANS & SUTHERLAND COMPUTER CORPORATION
P.O. Box 8700, 580 Arapeen Drive
Salt Lake City, Utah 84108

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:

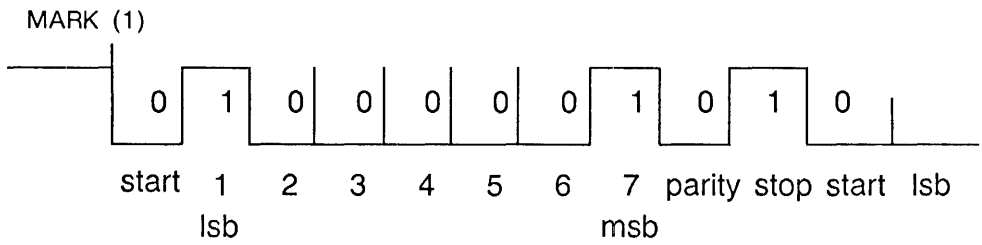
lsb						msb	
1	2	3	4	5	6	7	parity
1	1	1	0	0	0	1	0

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.)

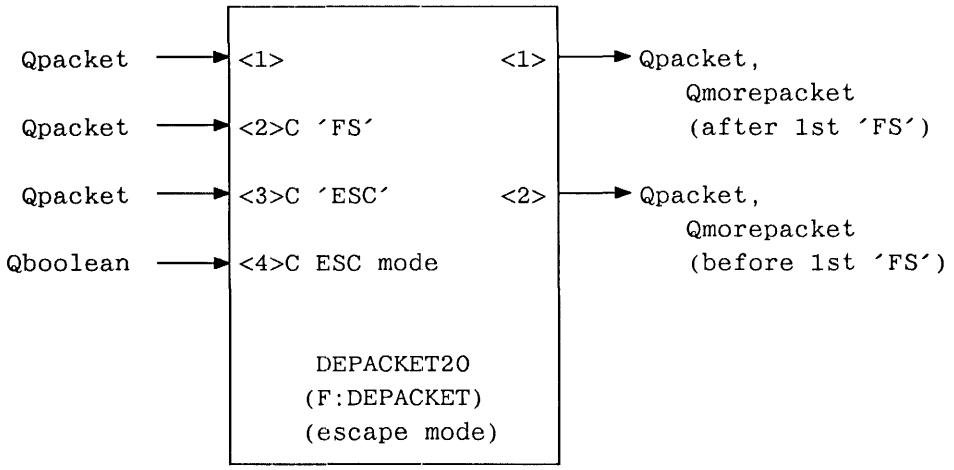
There are two instances of the F:DEPACKET function. The first, DEPACKET0, accepts all incoming bytes from the host on input <1>. It channels all incoming data through to output<2> until it sees the Start of Packet (SOP) character <ACK> (ACKNOWLEDGE — decimal character code 06, ASCII ↑F) that signifies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

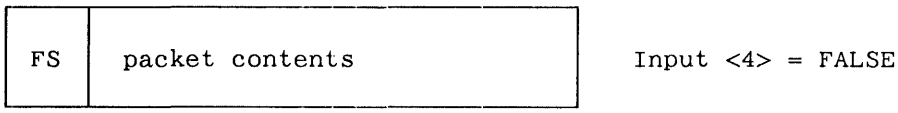
These instances of F:DEPACKET are described below. The characters that are used to signify SOP (<FS> and <ESC> characters) may be changed by the user by sending the new characters to the correct inputs of F:DEPACKET.

4.1.1 Escape Mode

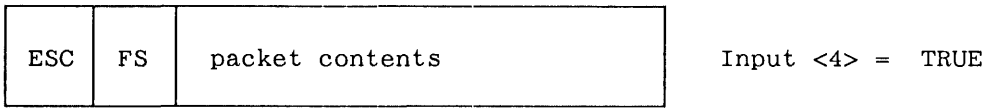
In escape mode, F:DEPACKET looks at every byte to see if it is a SOP character, which by default in escape mode is the ASCII Field Separator <FS> character, or an <ESC> character.



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.

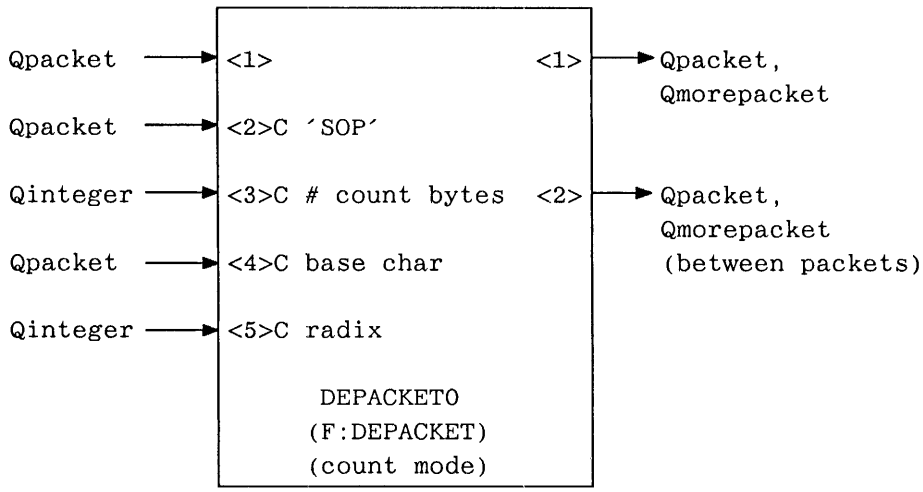
The ESC character is defined by a single character Qpacket on Input <3>. Output <1> outputs Qpacket and Qmorepackets of any messages after the first SOP control character is received. Output <2> outputs Qpackets and Qmorepackets of any messages before the first SOP control character is received. A Qpacket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

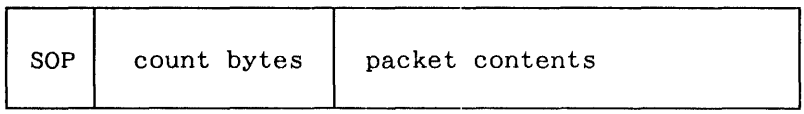
The routing path will be used for data transfer until the multiplexing function sees another SOP character, and a packet with another routing byte.

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:



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>DEPACKET0;
```

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.

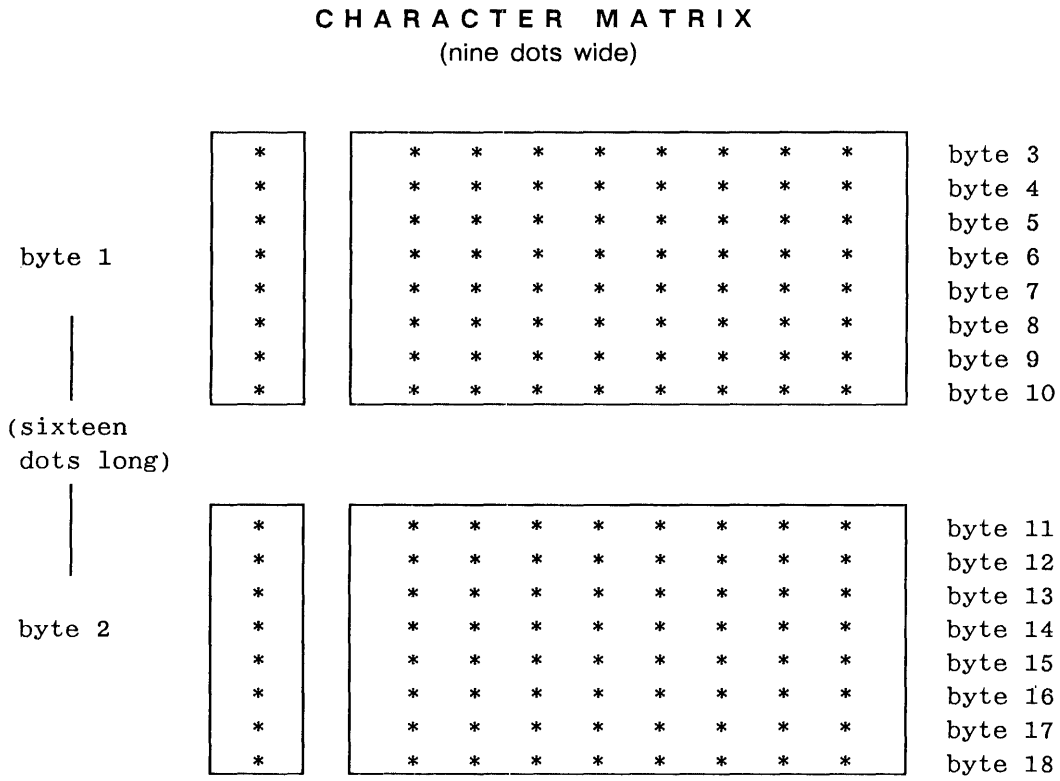
If the GSRs are not used, the user will need to have some understanding of how Programmed Symbols work and how the 3274 sends the symbols to the 3278 to understand how the WSF data buffers are built. A detailed description of Programmed Symbols and their use to transfer graphics data is provided below.

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.

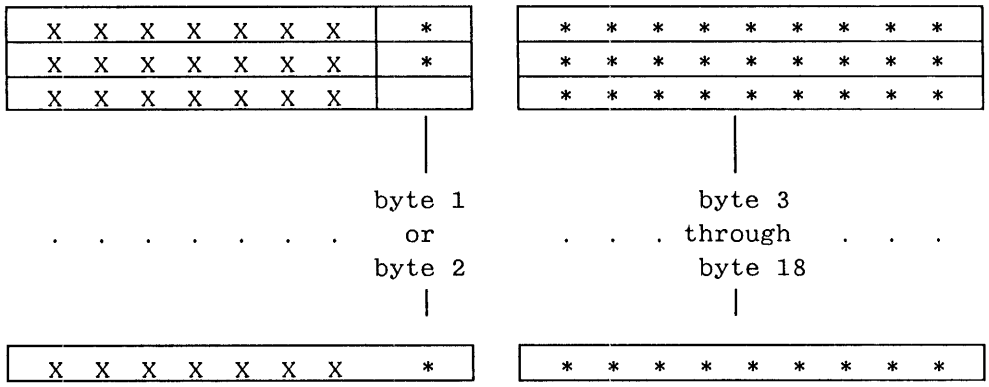


The data that describe the matrix are placed in a WSF buffer in the following order.

```
  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



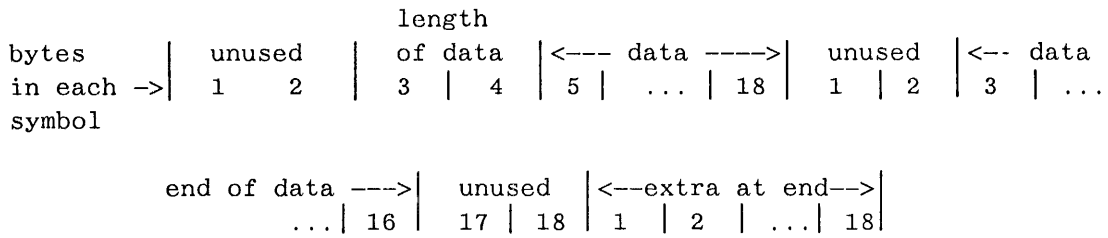
The PS 390 receives graphics data passed to it from the 3274 in the format shown above. In order for the PS 390 to avoid the difficulty of reassembling the bytes received, it simply discards the first byte of each of the sixteen two-byte pairs for each programmed symbol. This means that the first two 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.



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

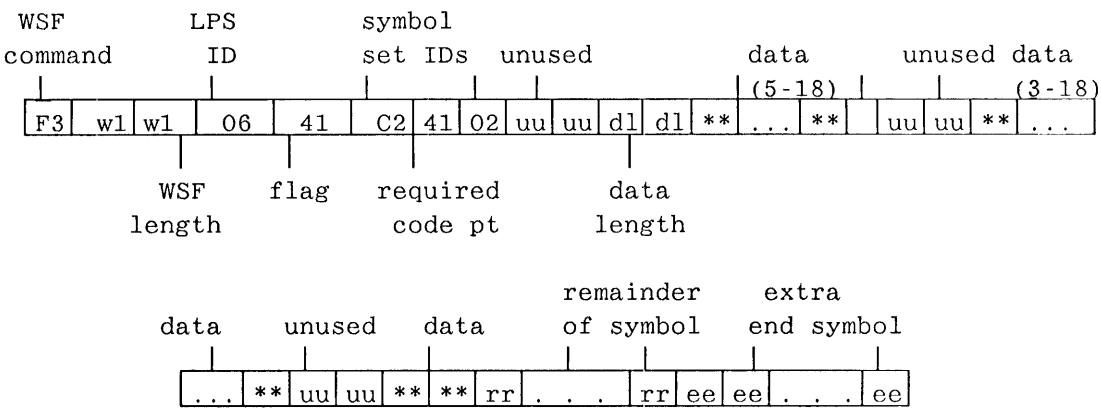
The Load Programmed Symbols option of the WSF command that is used to load the symbols described in preceding paragraphs is invoked by inserting control information after the WSF command code and before the programmed symbols.

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.

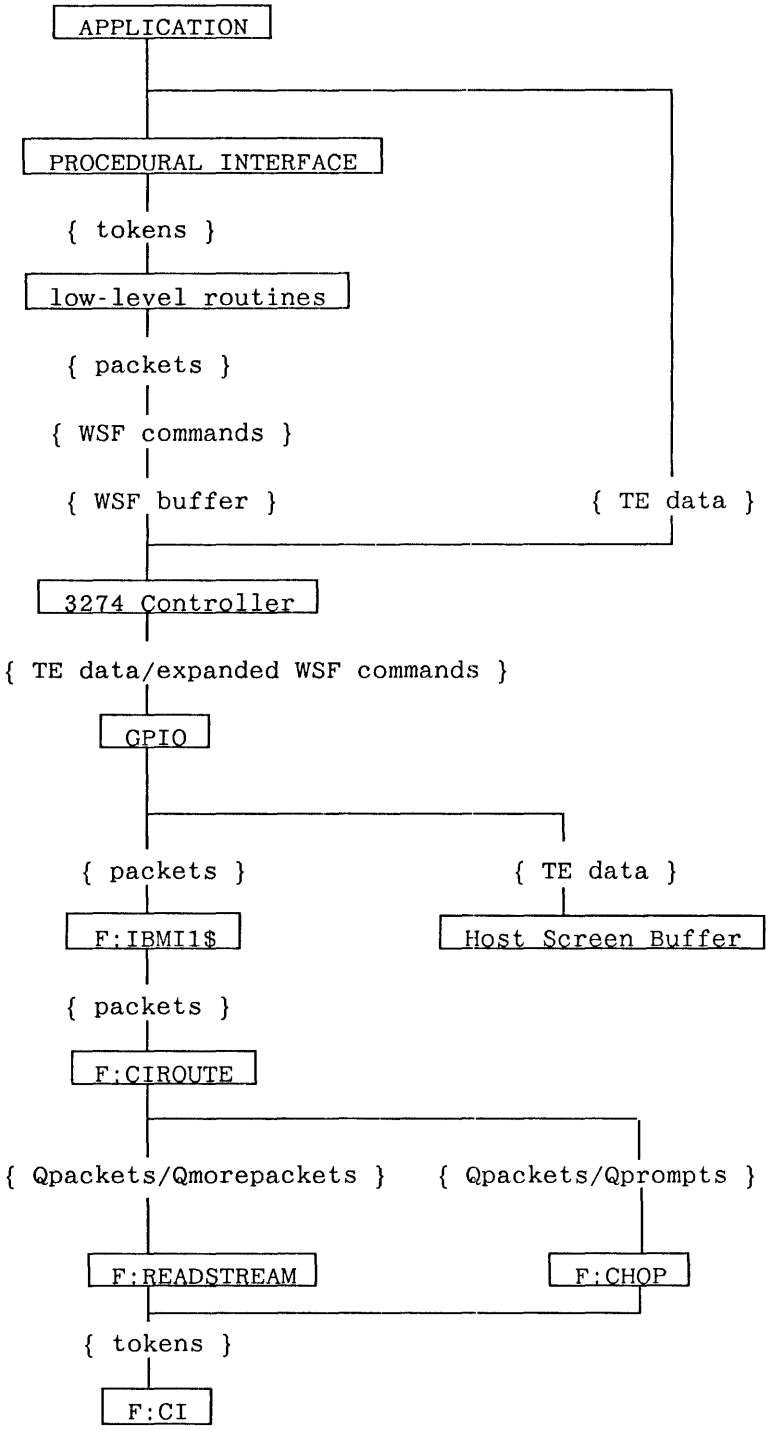
164 — Programmed Symbols

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;
```

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

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

CIROUTEO 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.

RM8. SYSTEM FUNCTION NETWORK

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.

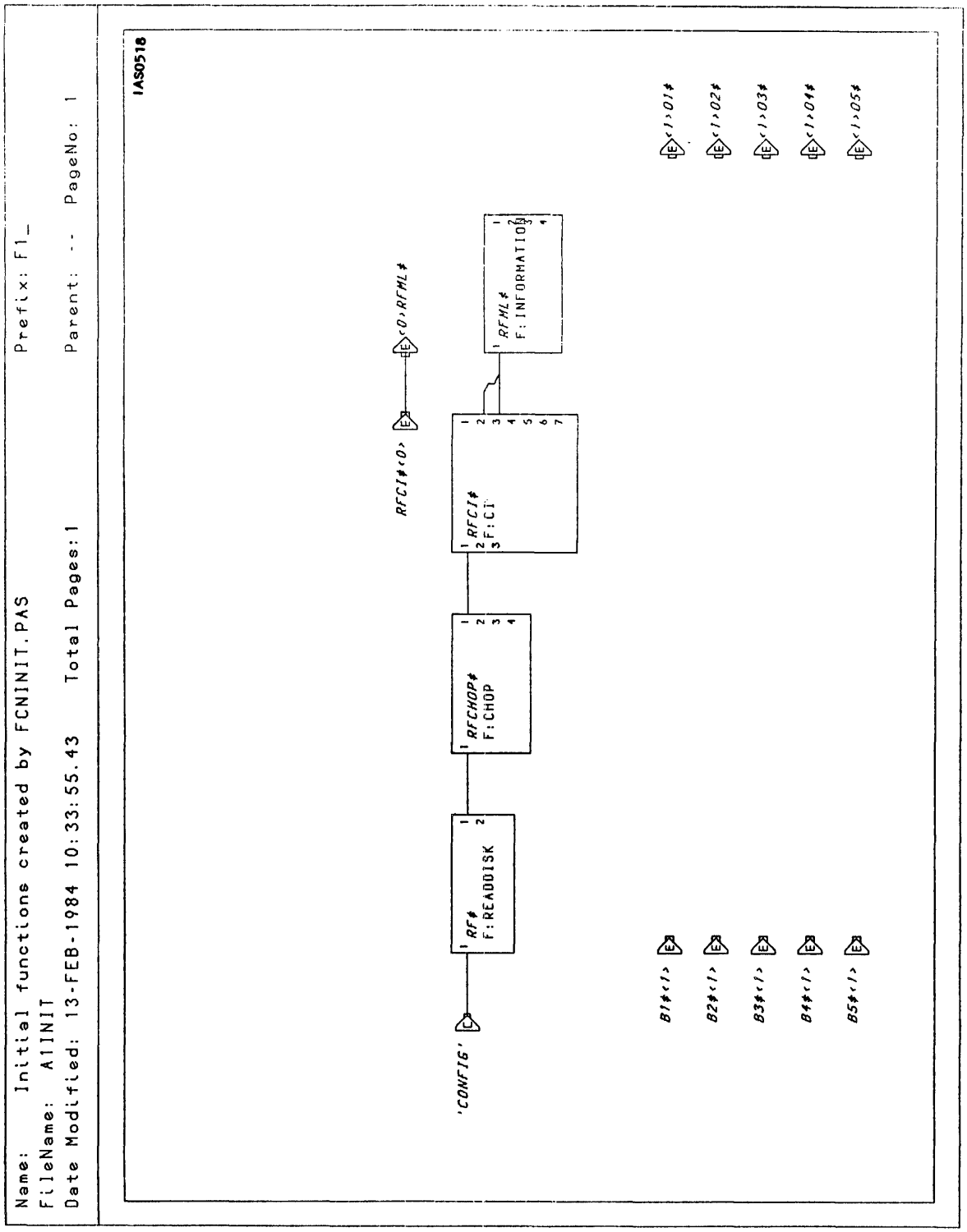


Figure 8-1. PS 390 Initial Read Floppy Network Created

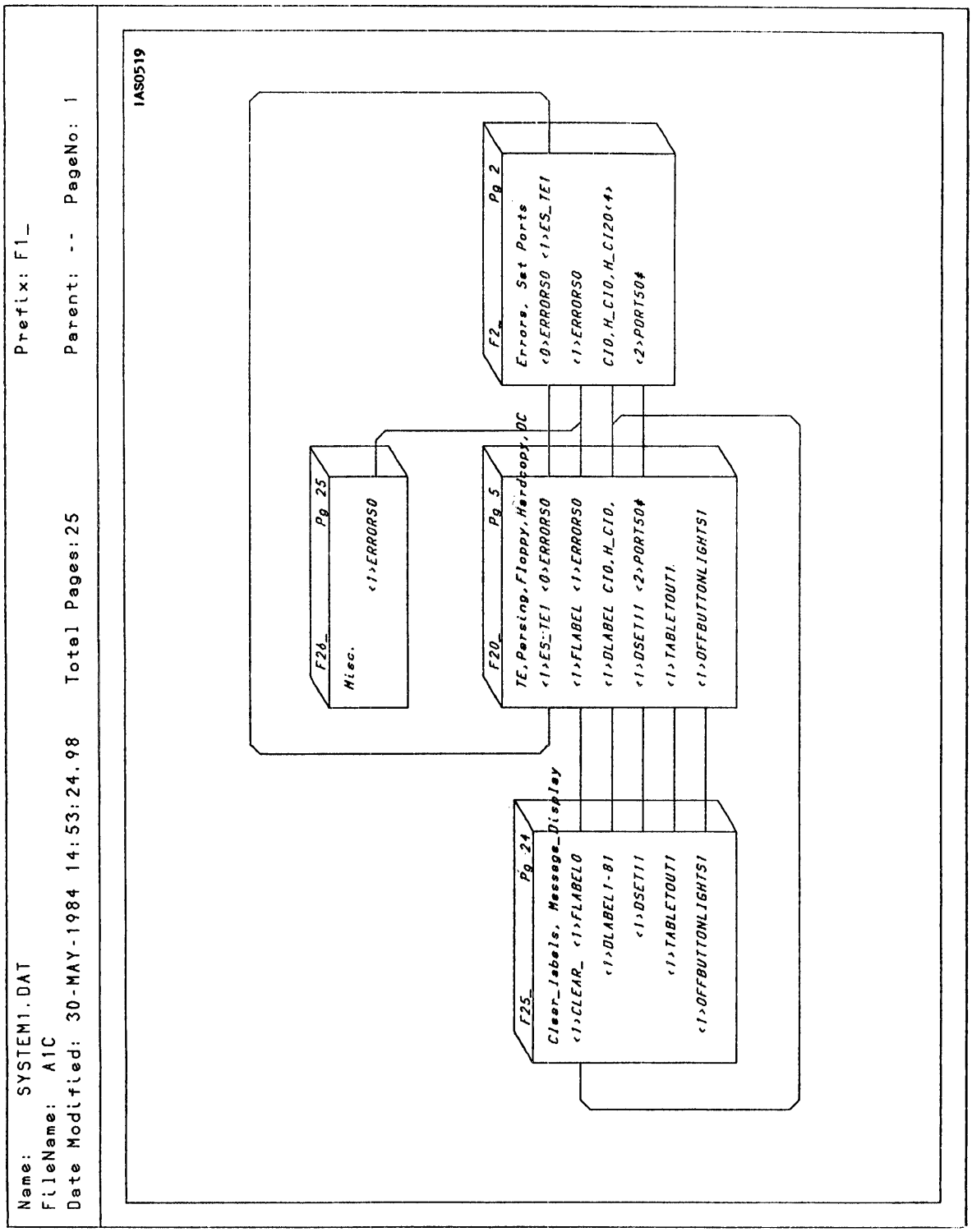


Figure 8-2. PS 390 Host Input Data Flow (RS-232 Interface)

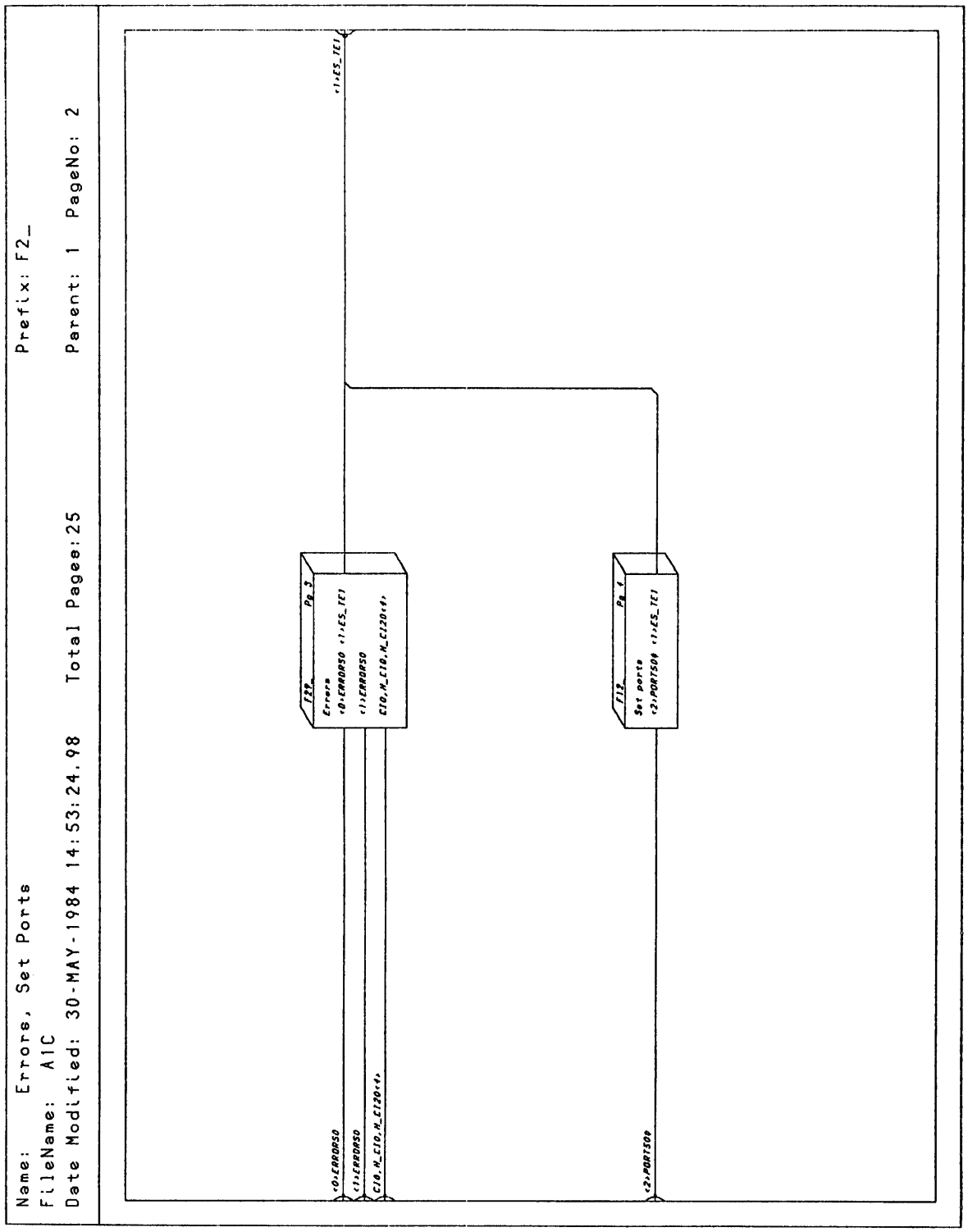


Figure 8-3. PS 390 Host Input Data Flow (RS-232 Interface)

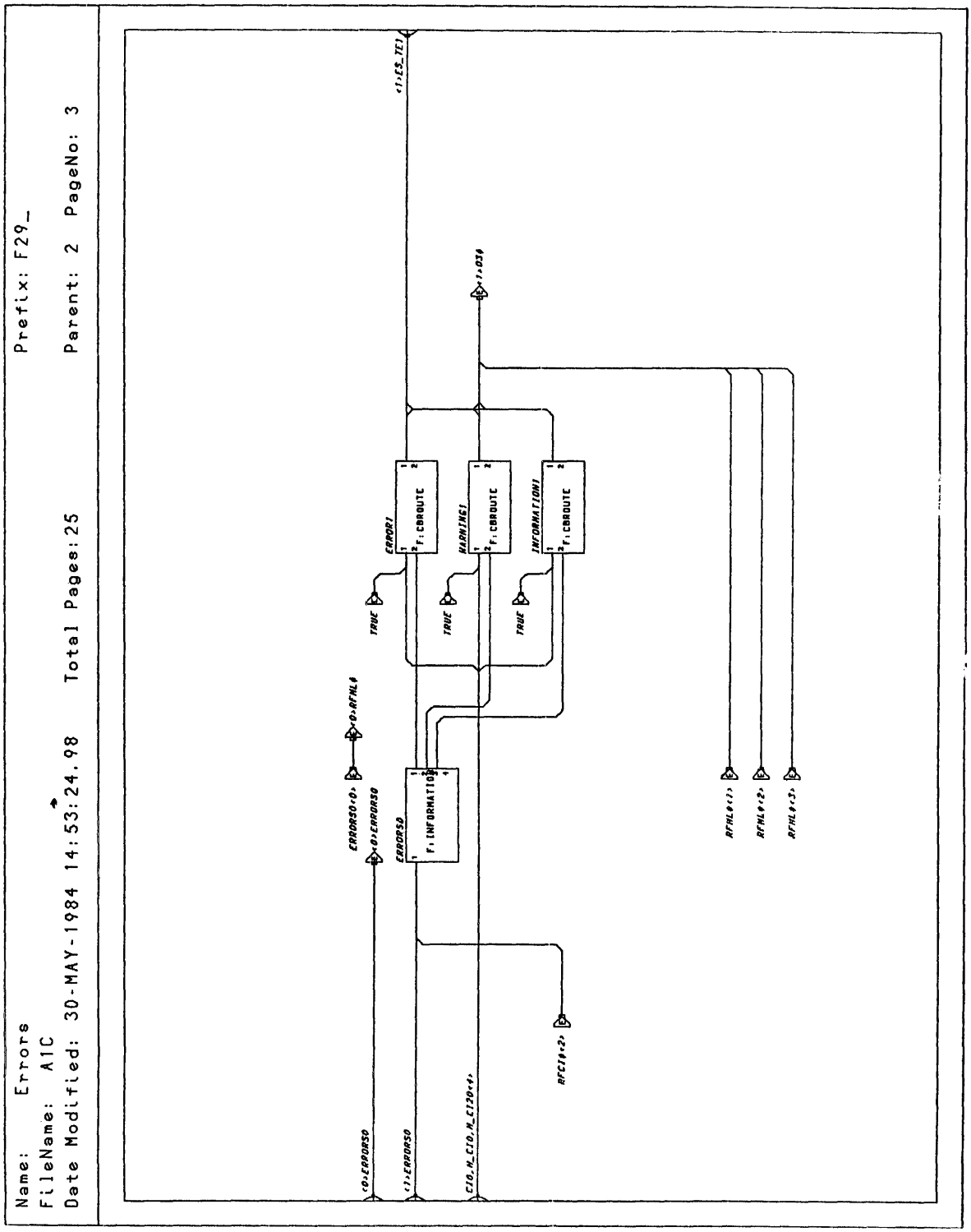


Figure 8-4. PS 390 Host Input Data Flow (RS-232 Interface)

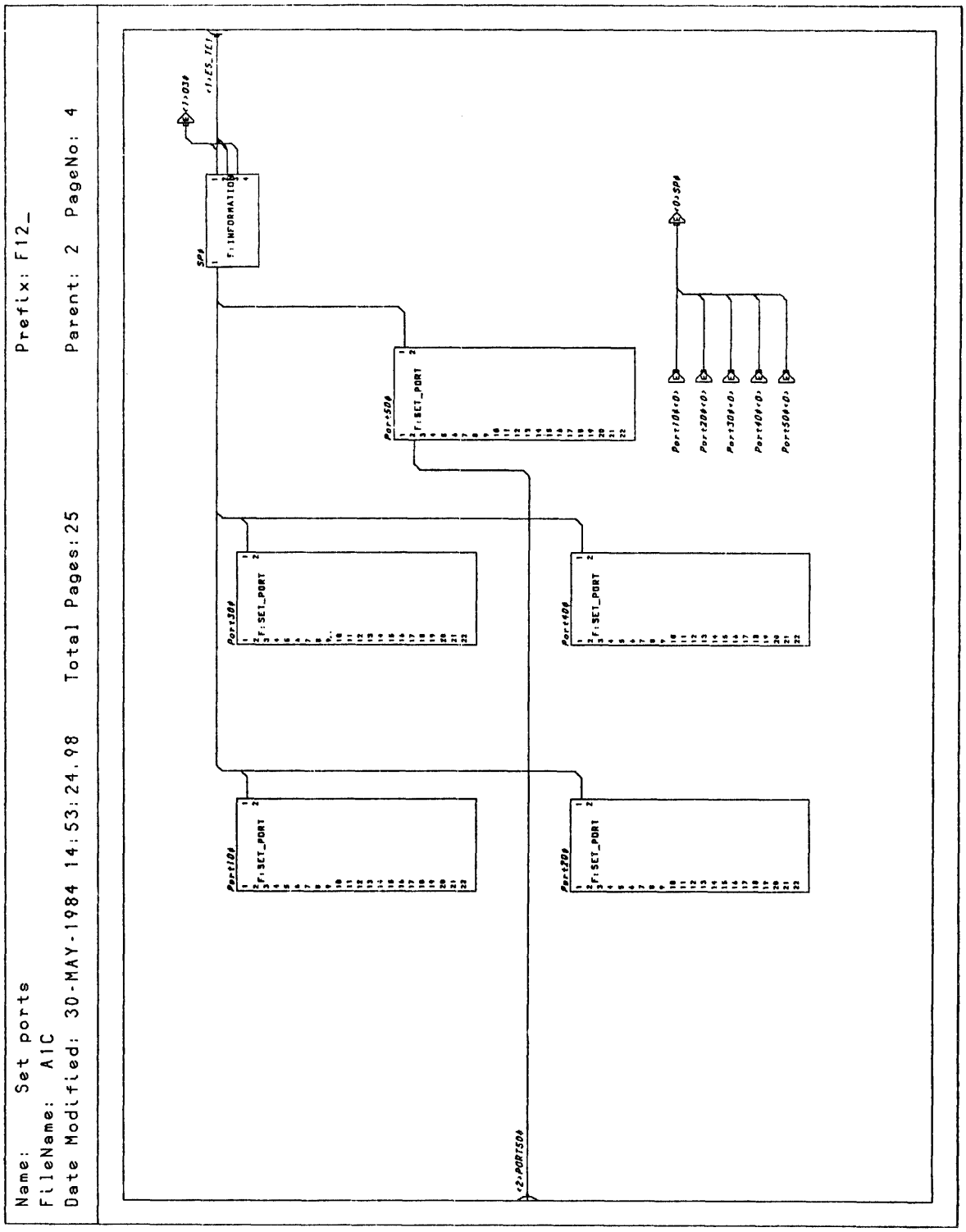


Figure 8-5. PS 390 Host Input Data Flow (RS-232 Interface)

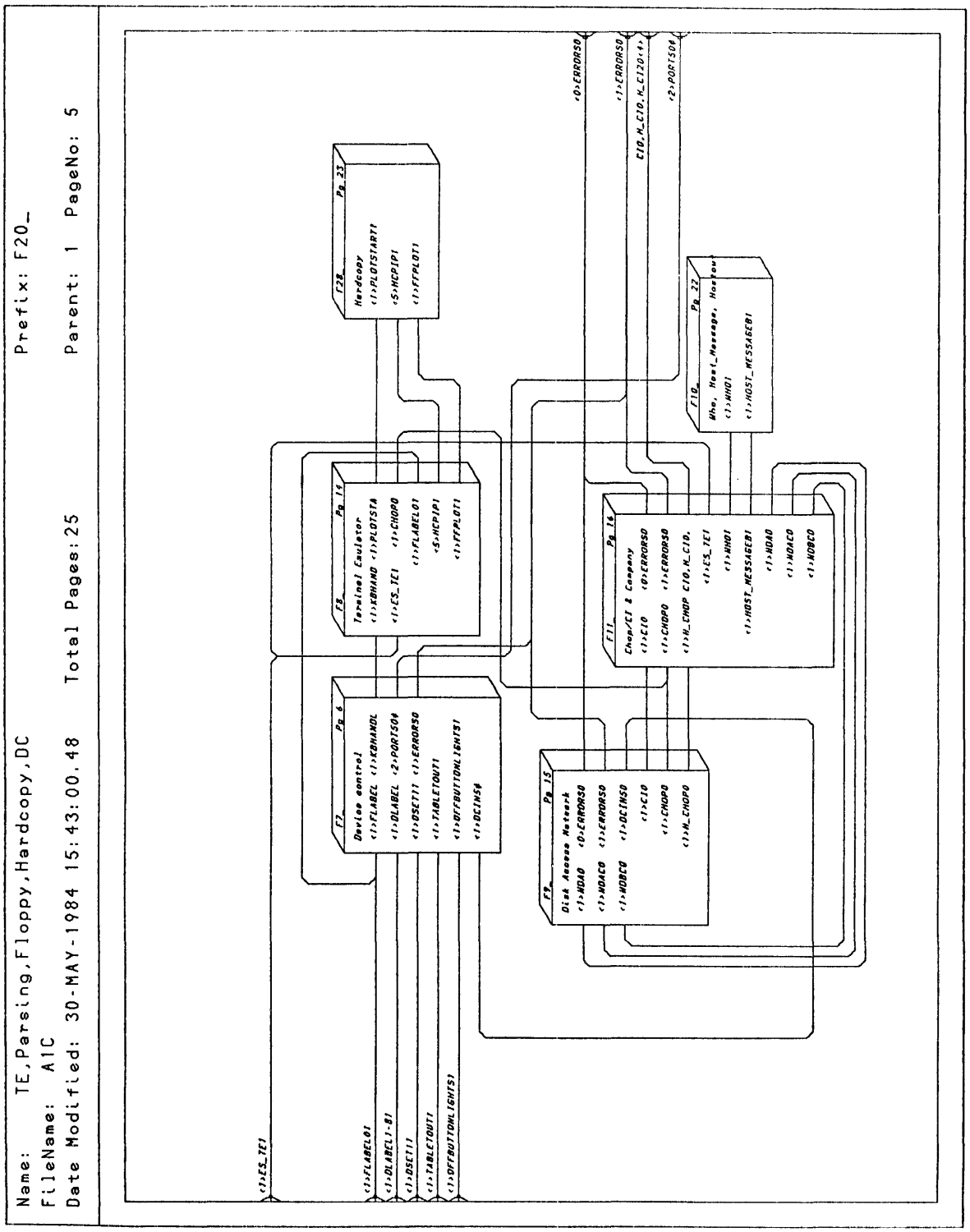


Figure 8-6. PS 390 Host Input Data Flow (RS-232 Interface)

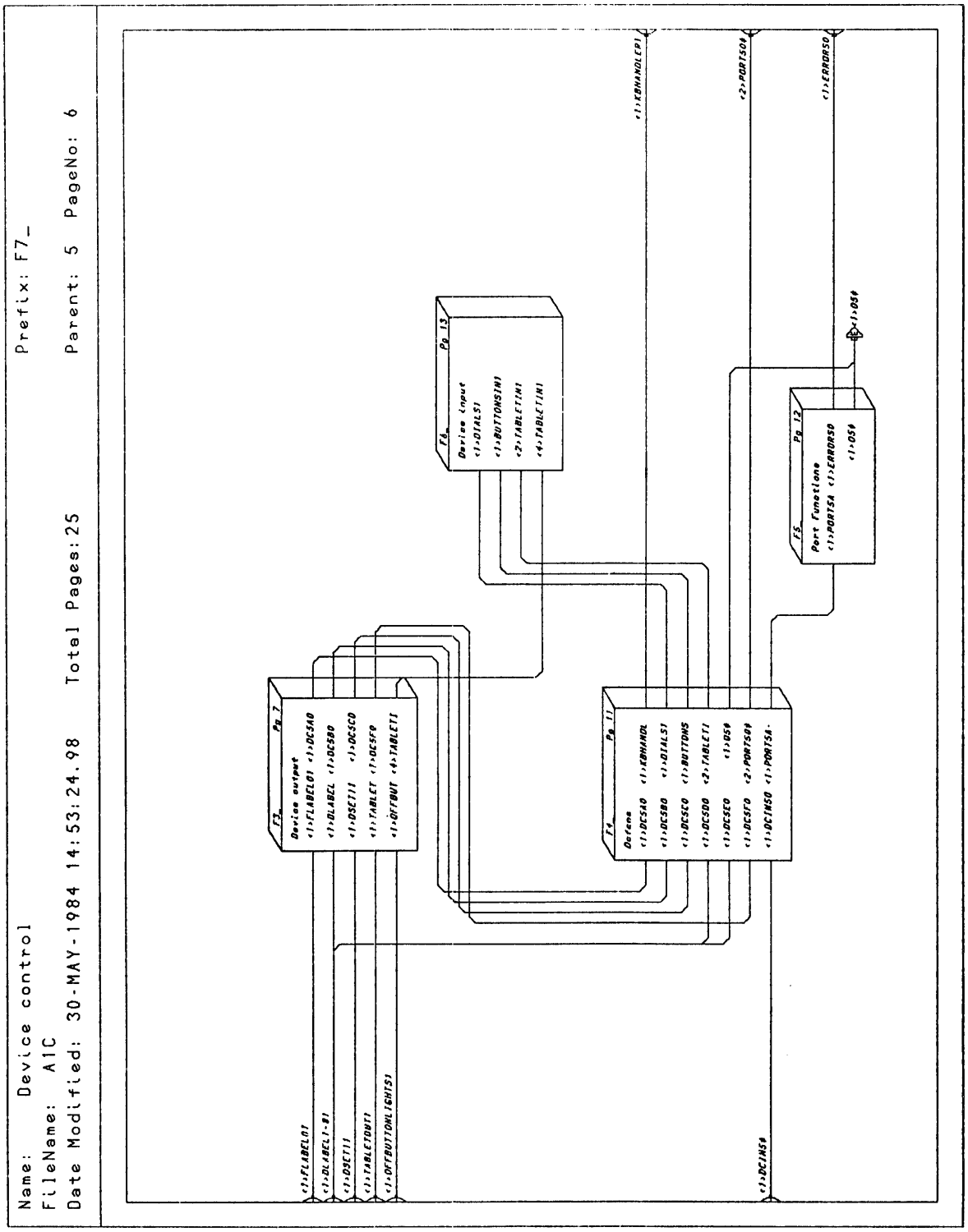


Figure 8-7. PS 390 Host Input Data Flow (RS-232 Interface)

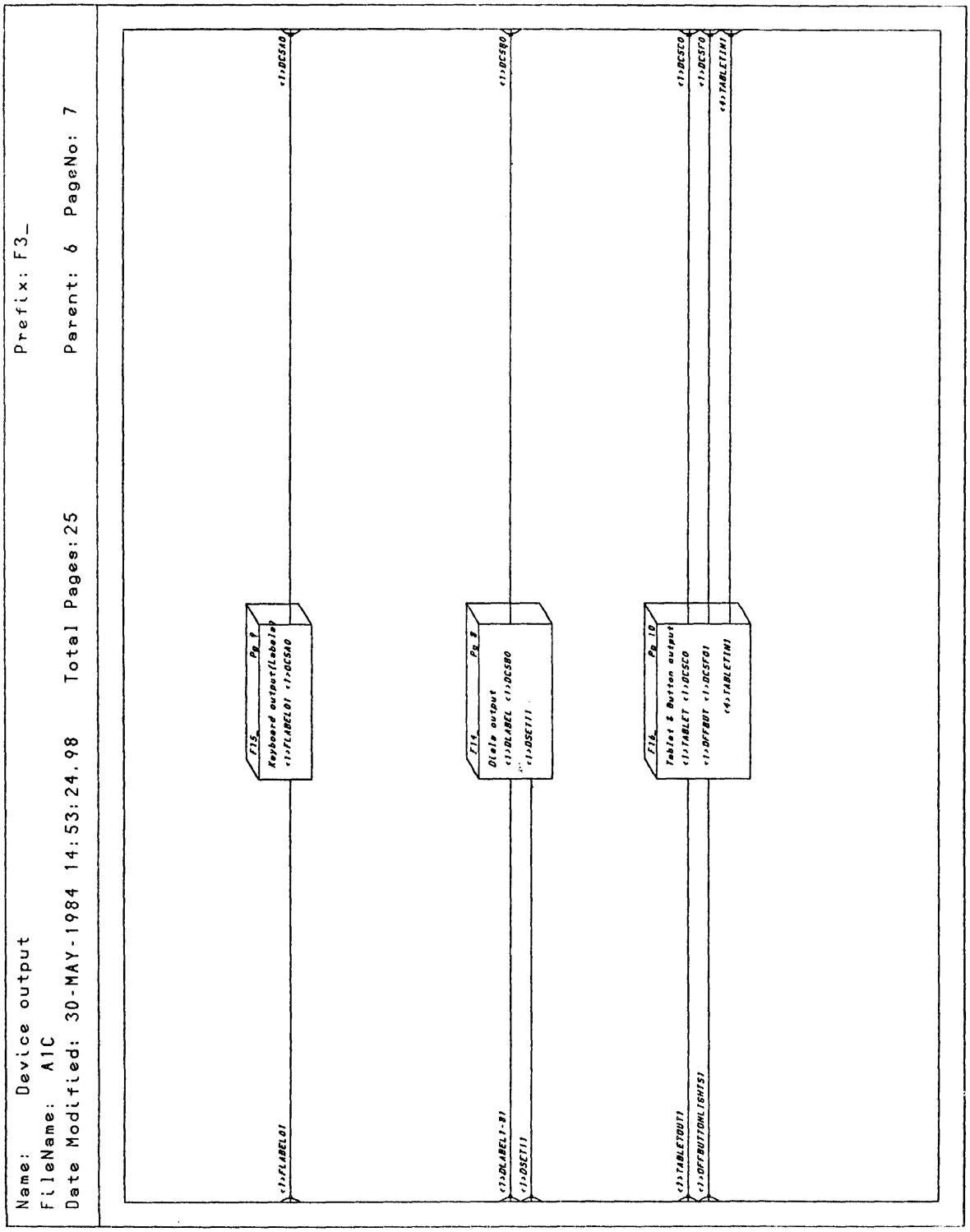


Figure 8-8. PS 390 Host Input Data Flow (RS-232 Interface)

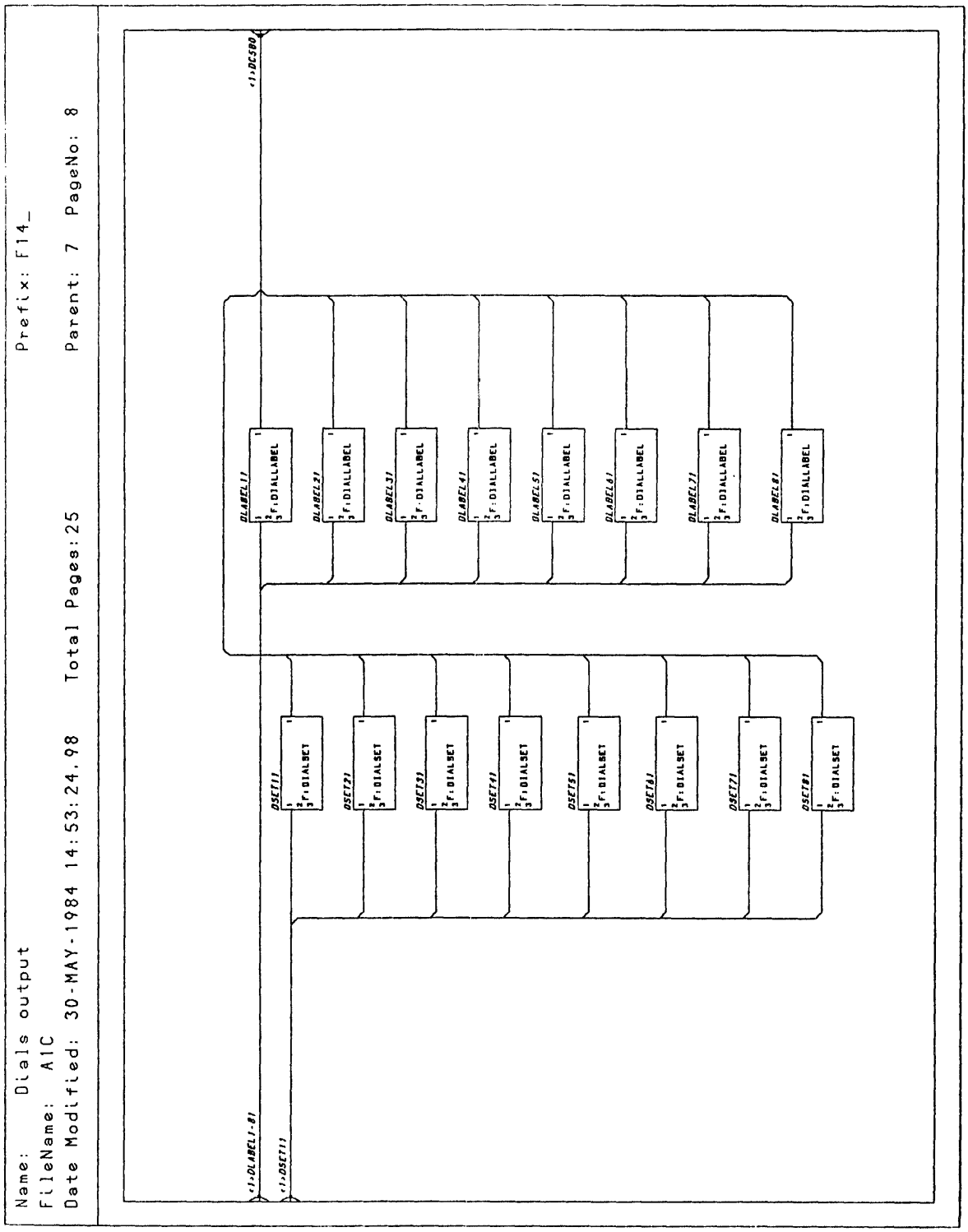


Figure 8-9. PS 390 Host Input Data Flow (RS-232 Interface)

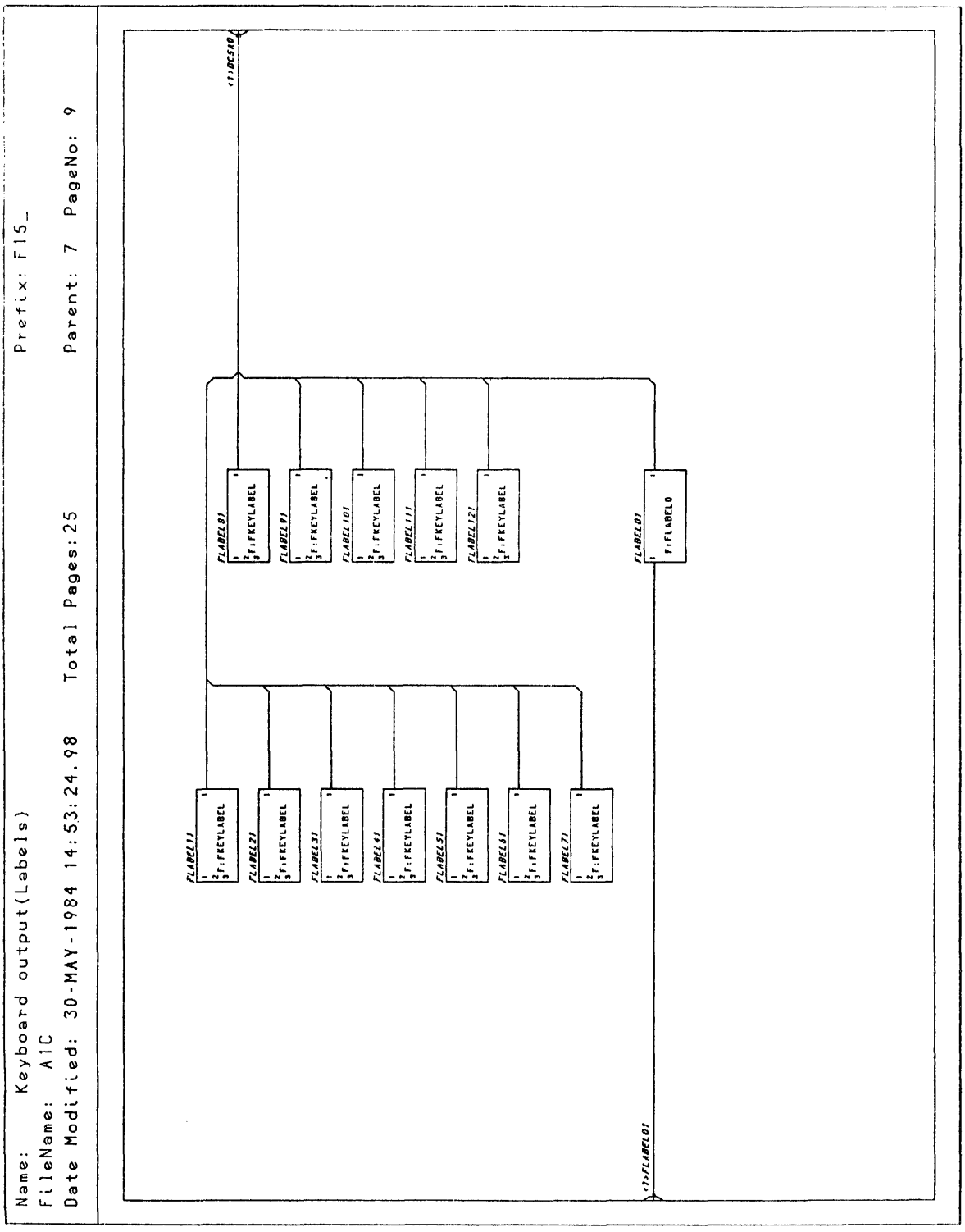


Figure 8-10. PS 390 Host Input Data Flow (RS-232 Interface)

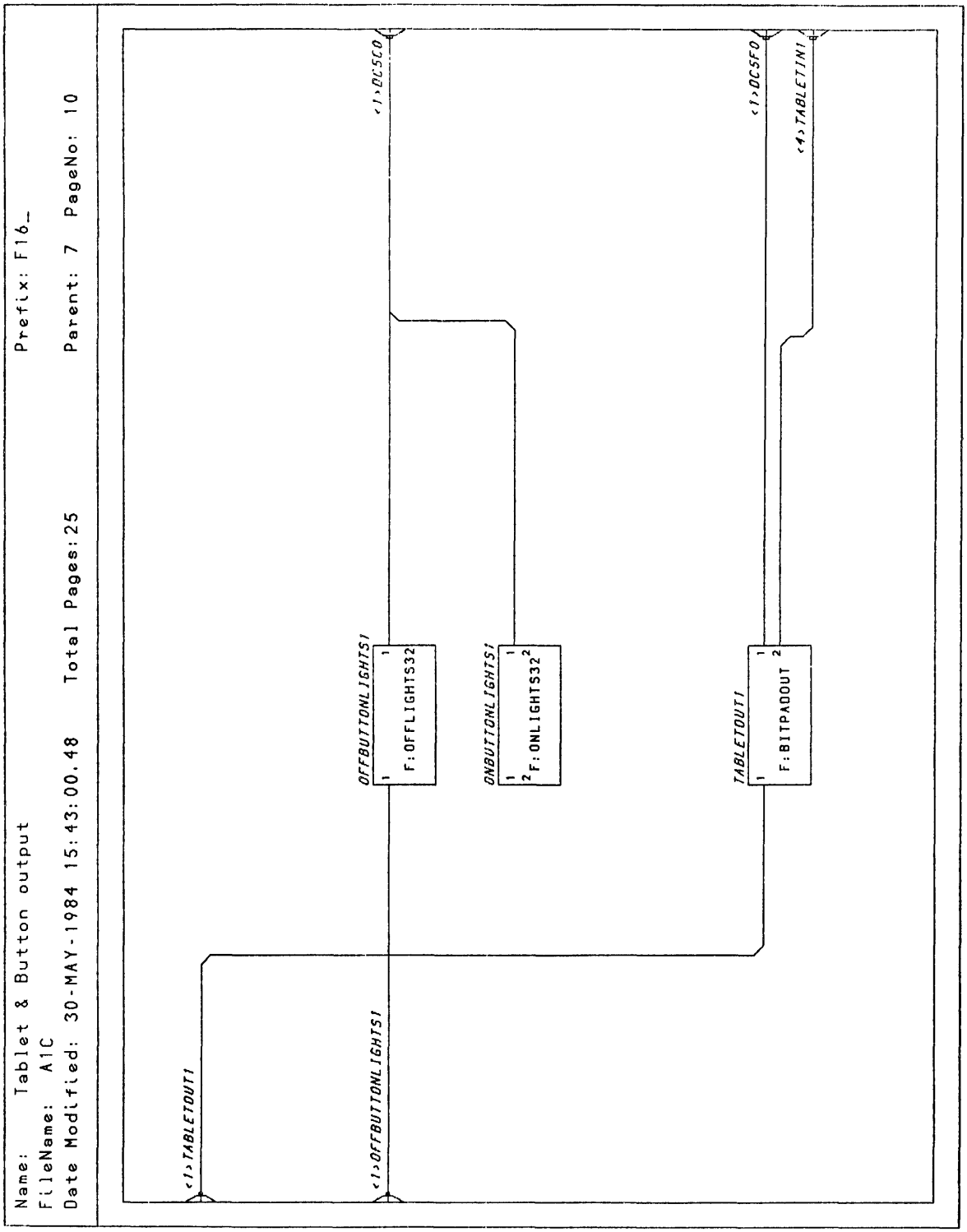


Figure 8-11. PS 390 Host Input Data Flow (RS-232 Interface)

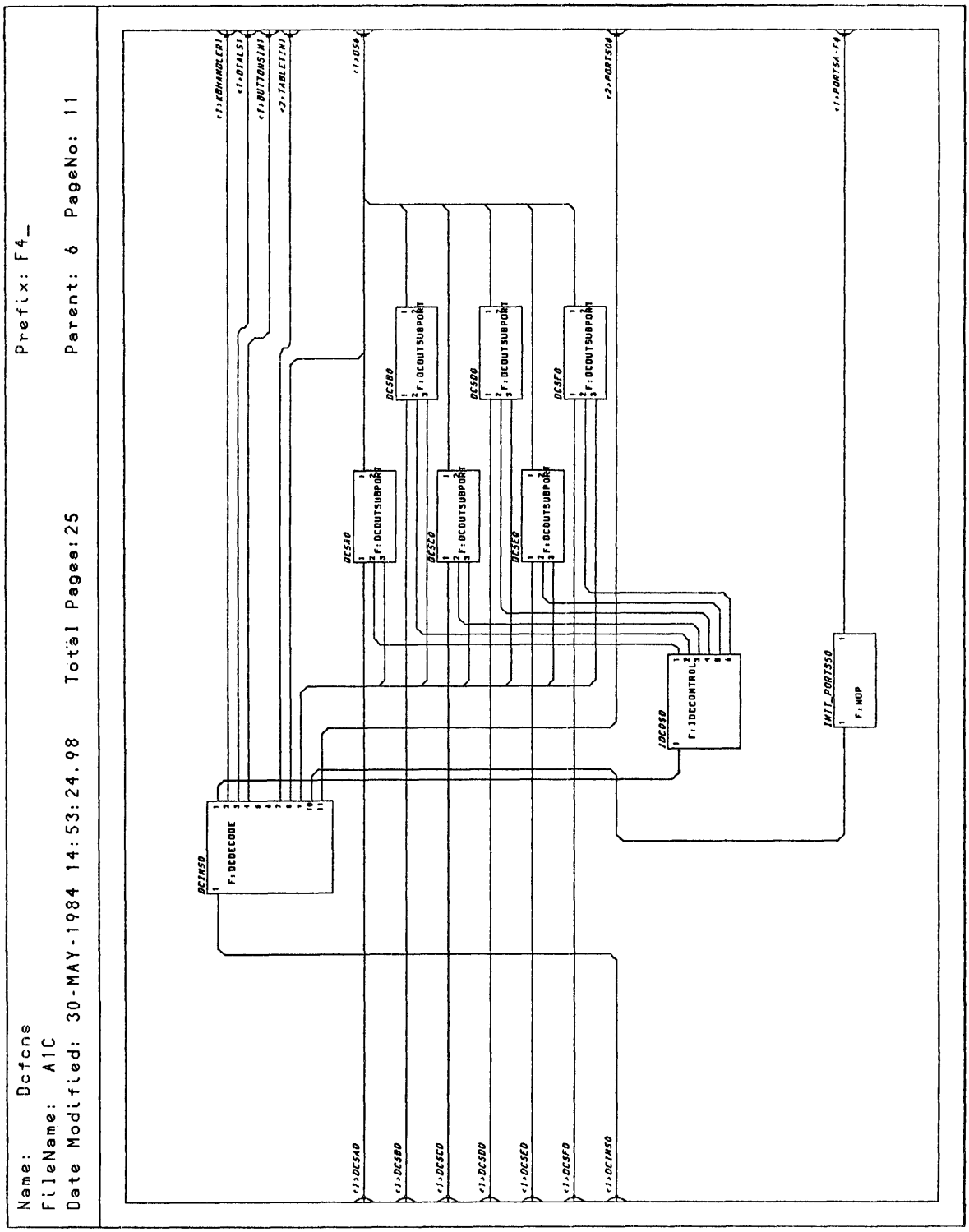


Figure 8-12. PS 390 Host Input Data Flow (RS-232 Interface)

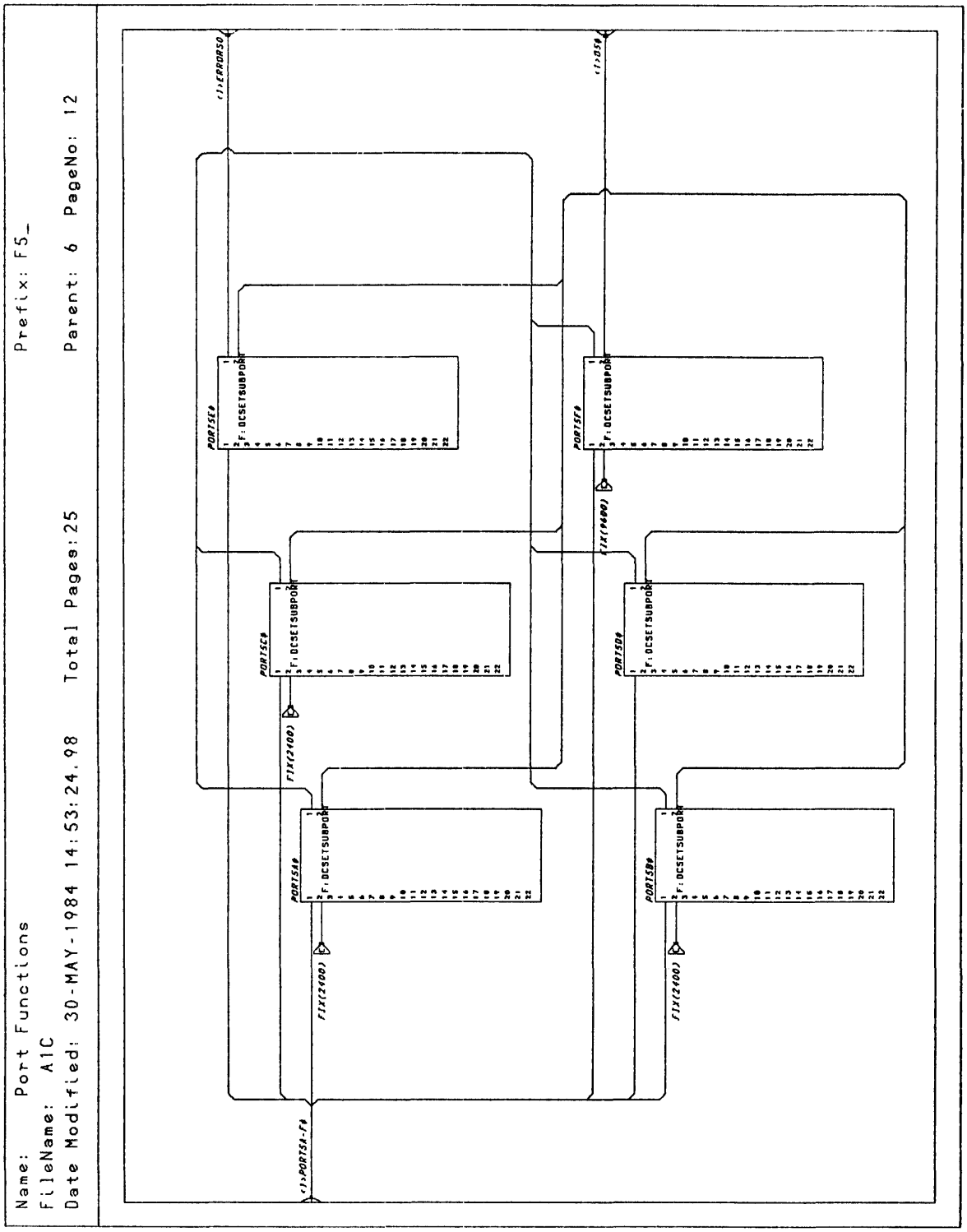


Figure 8-13. PS 390 Host Input Data Flow (RS-232 Interface)

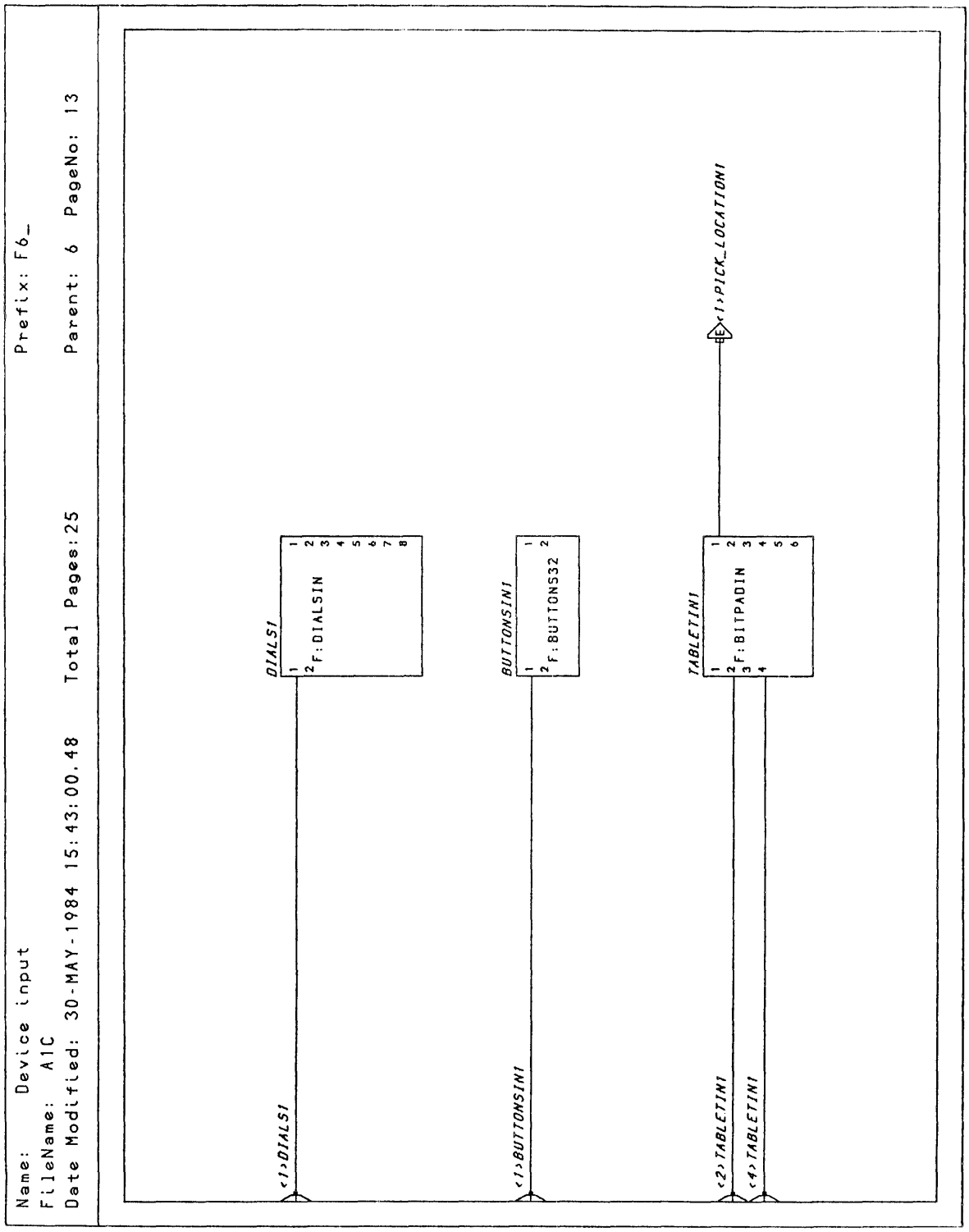


Figure 8-14. PS 390 Host Input Data Flow (RS-232 Interface)

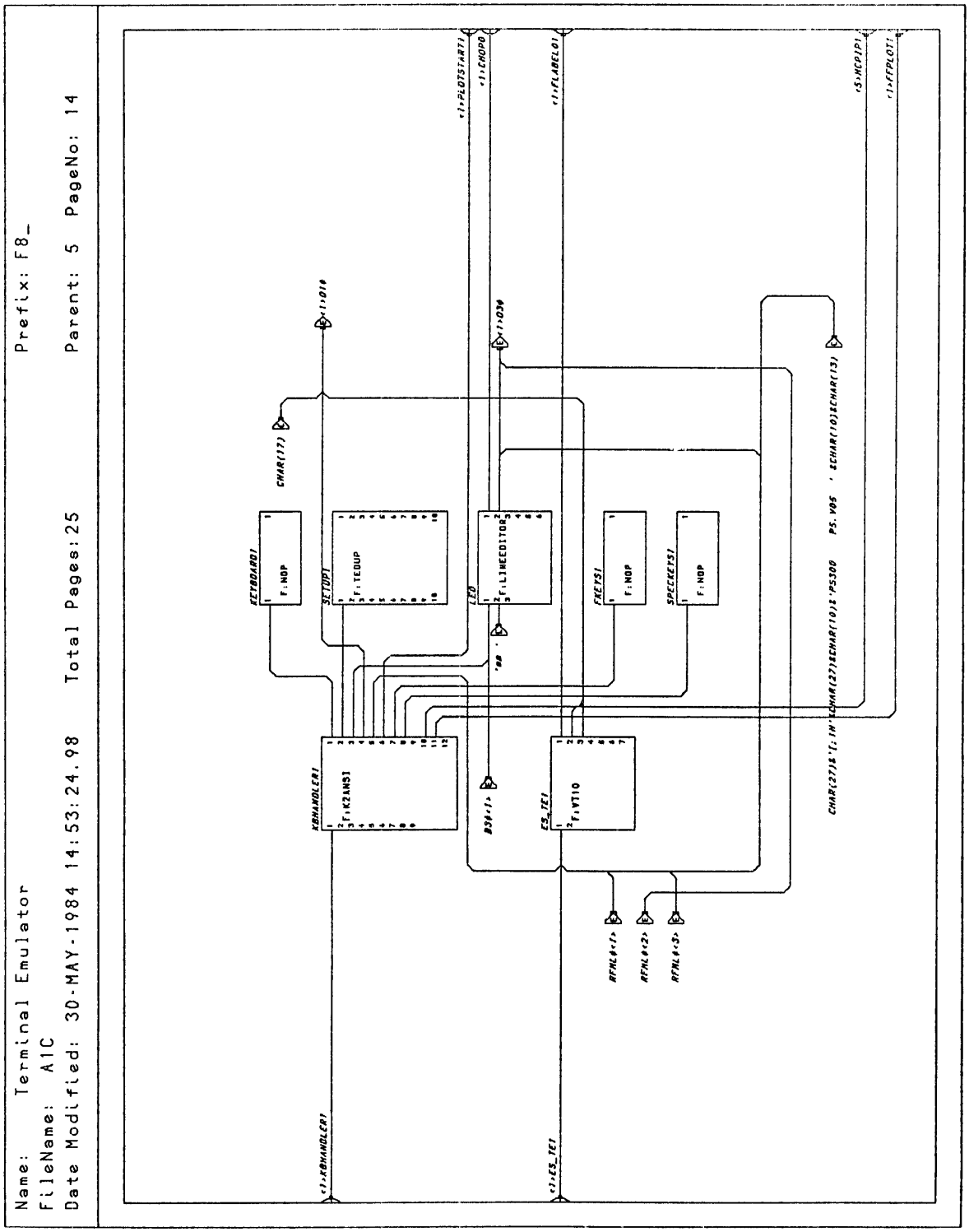


Figure 8-15. PS 390 Host Input Data Flow (RS-232 Interface)

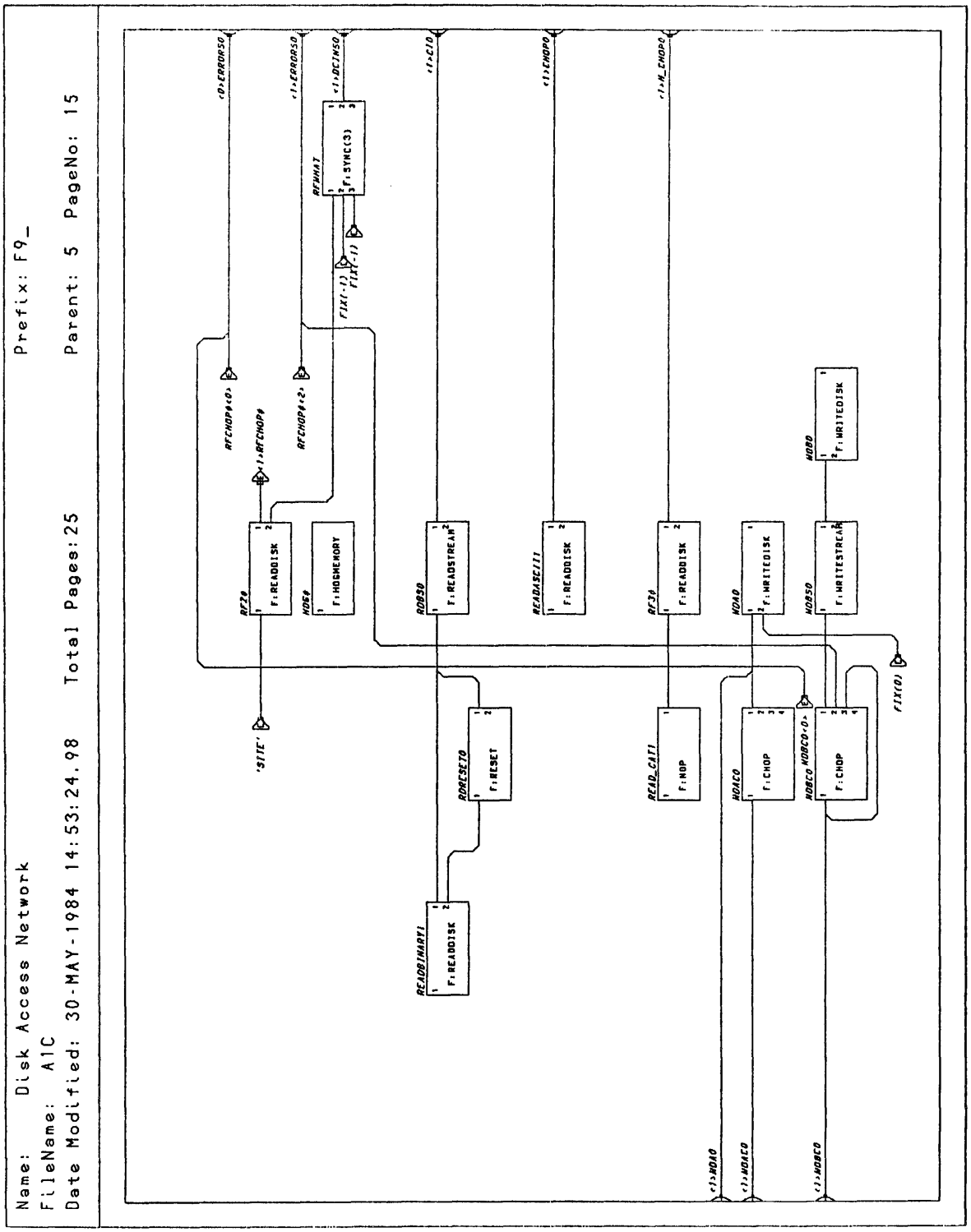


Figure 8-16. PS 390 Host Input Data Flow (RS-232 Interface)

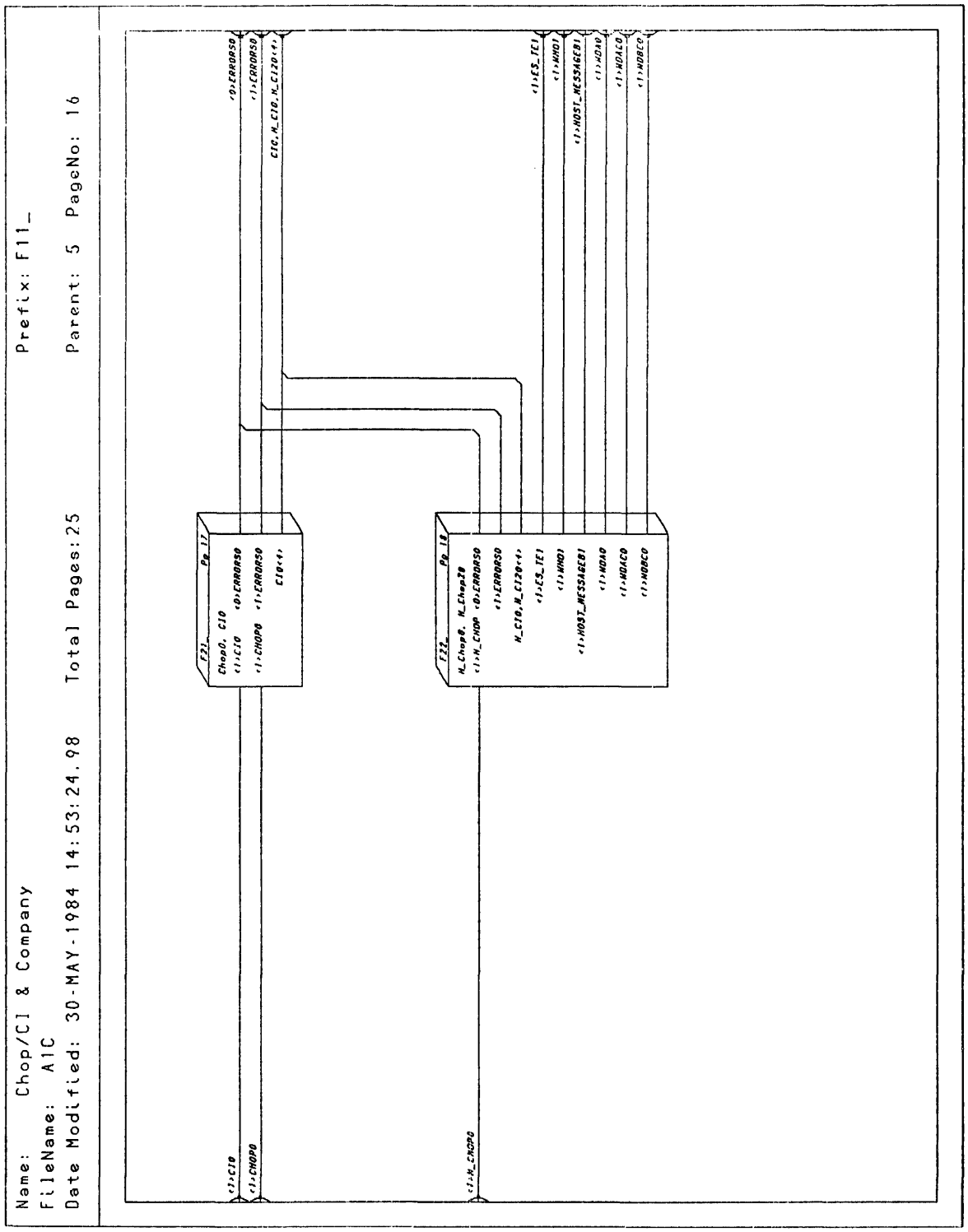


Figure 8-17. PS 390 Host Input Data Flow (RS-232 Interface)

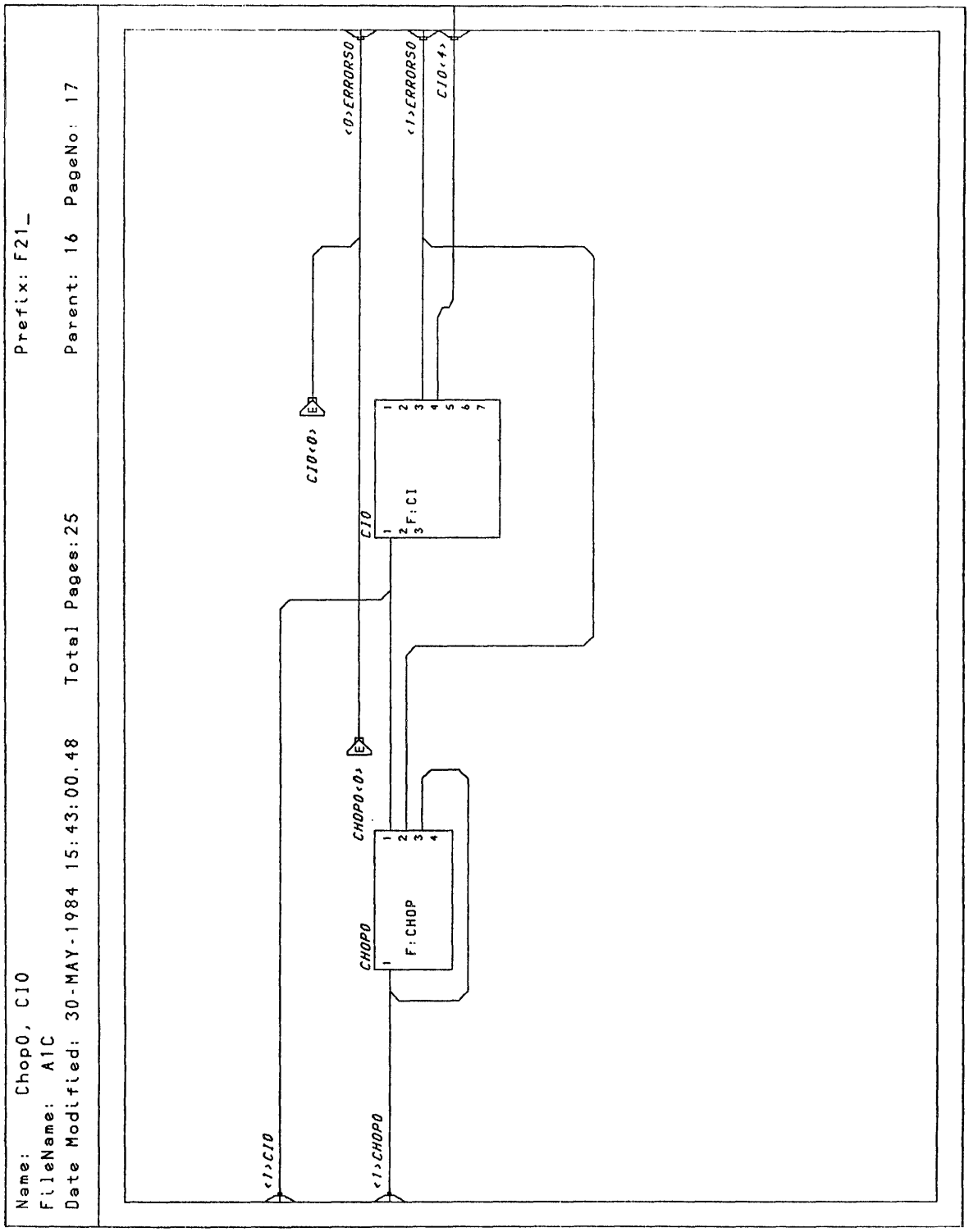


Figure 8-18. PS 390 Host Input Data Flow (RS-232 Interface)

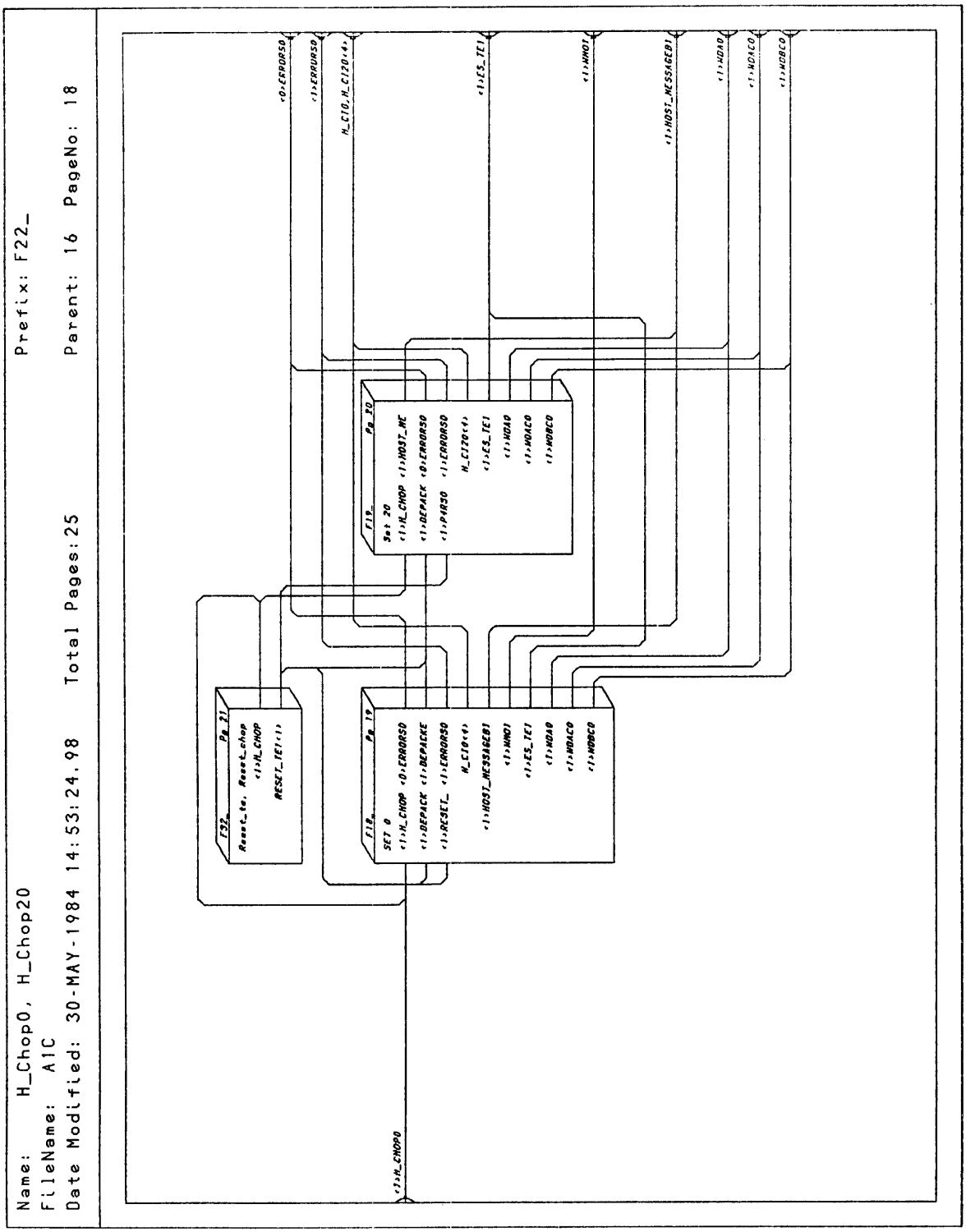


Figure 8-19. PS 390 Host Input Data Flow (RS-232 Interface)

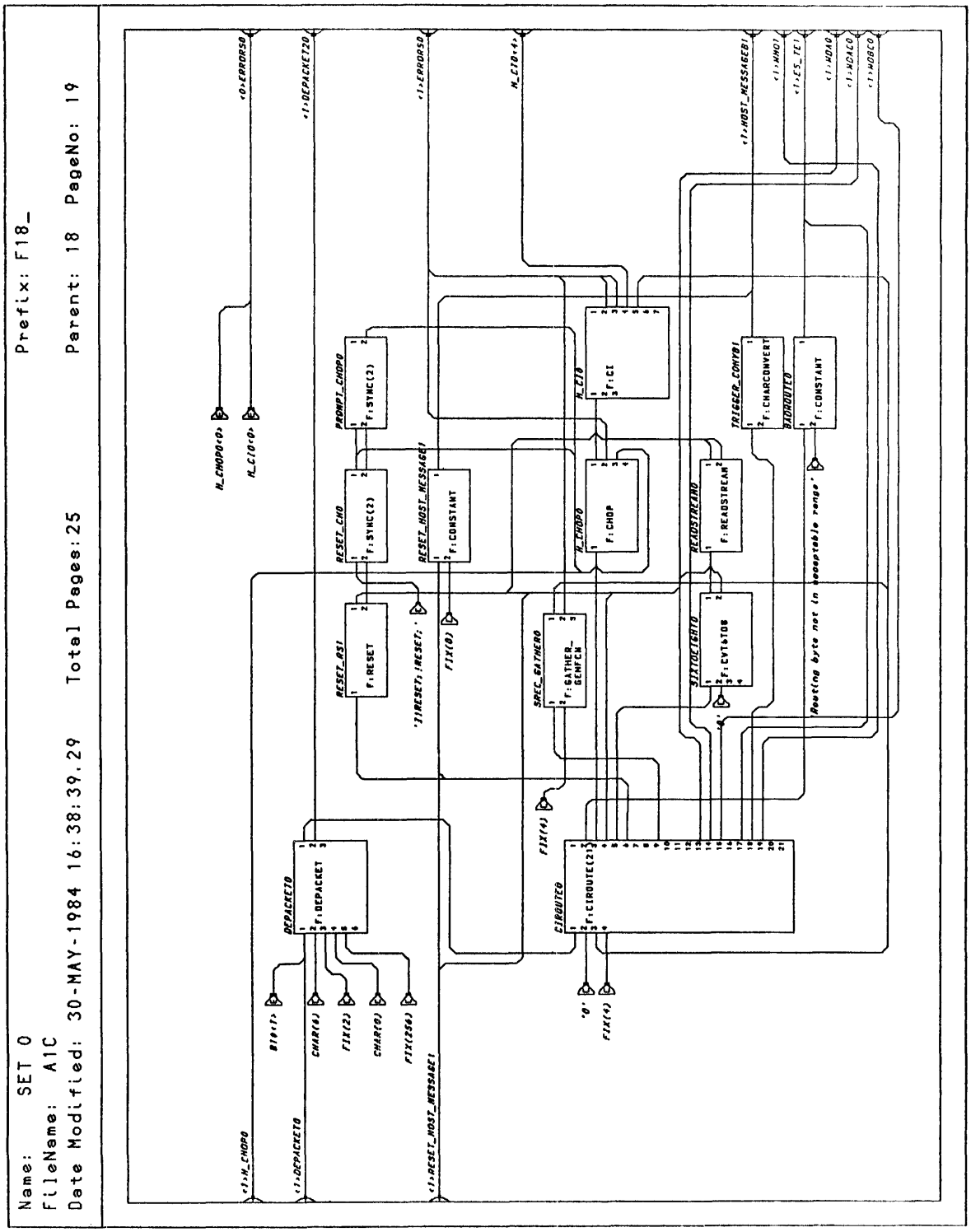


Figure 8-20. PS 390 Host Input Data Flow (RS-232 Interface)

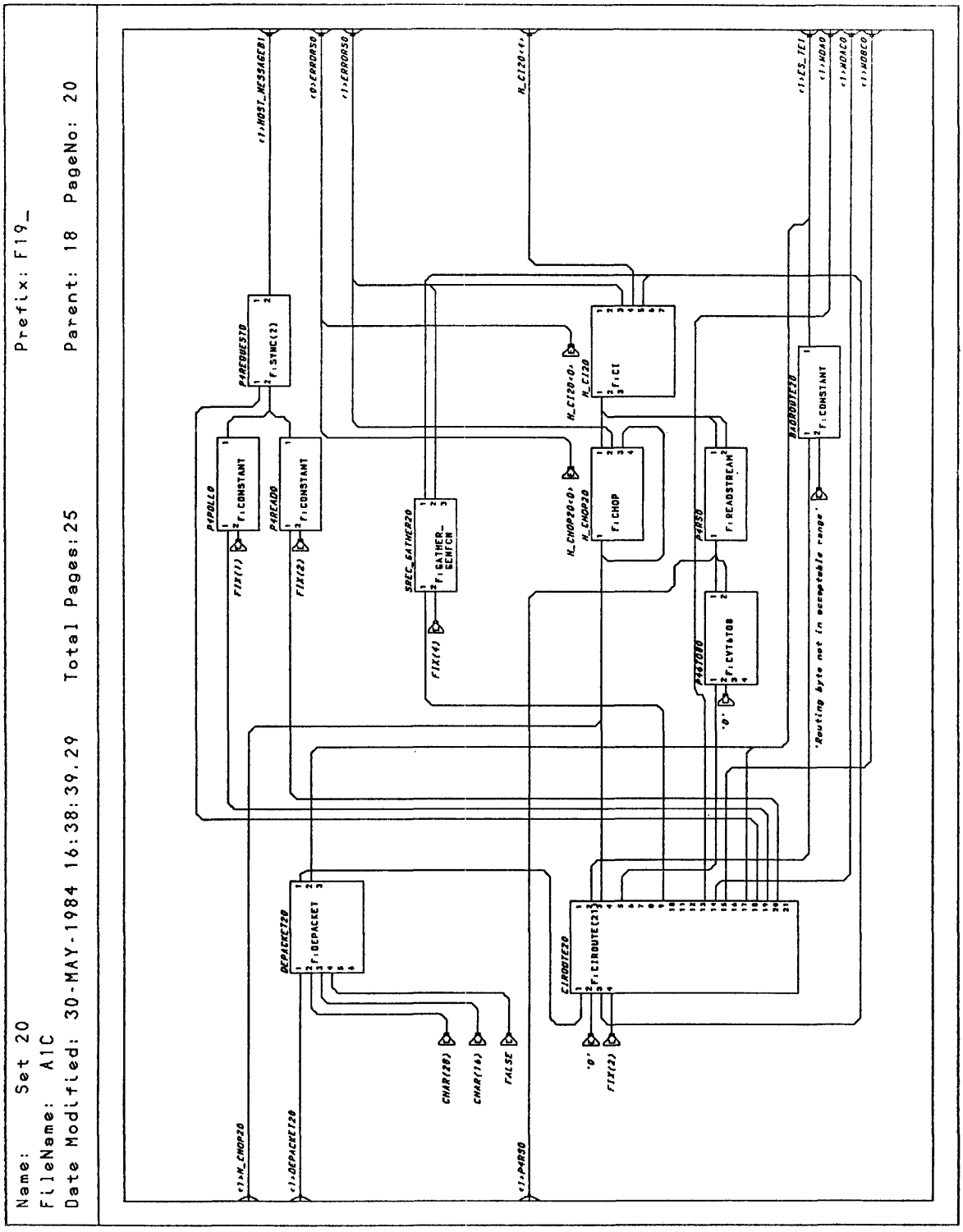


Figure 8-21. PS 390 Host Input Data Flow (RS-232 Interface)

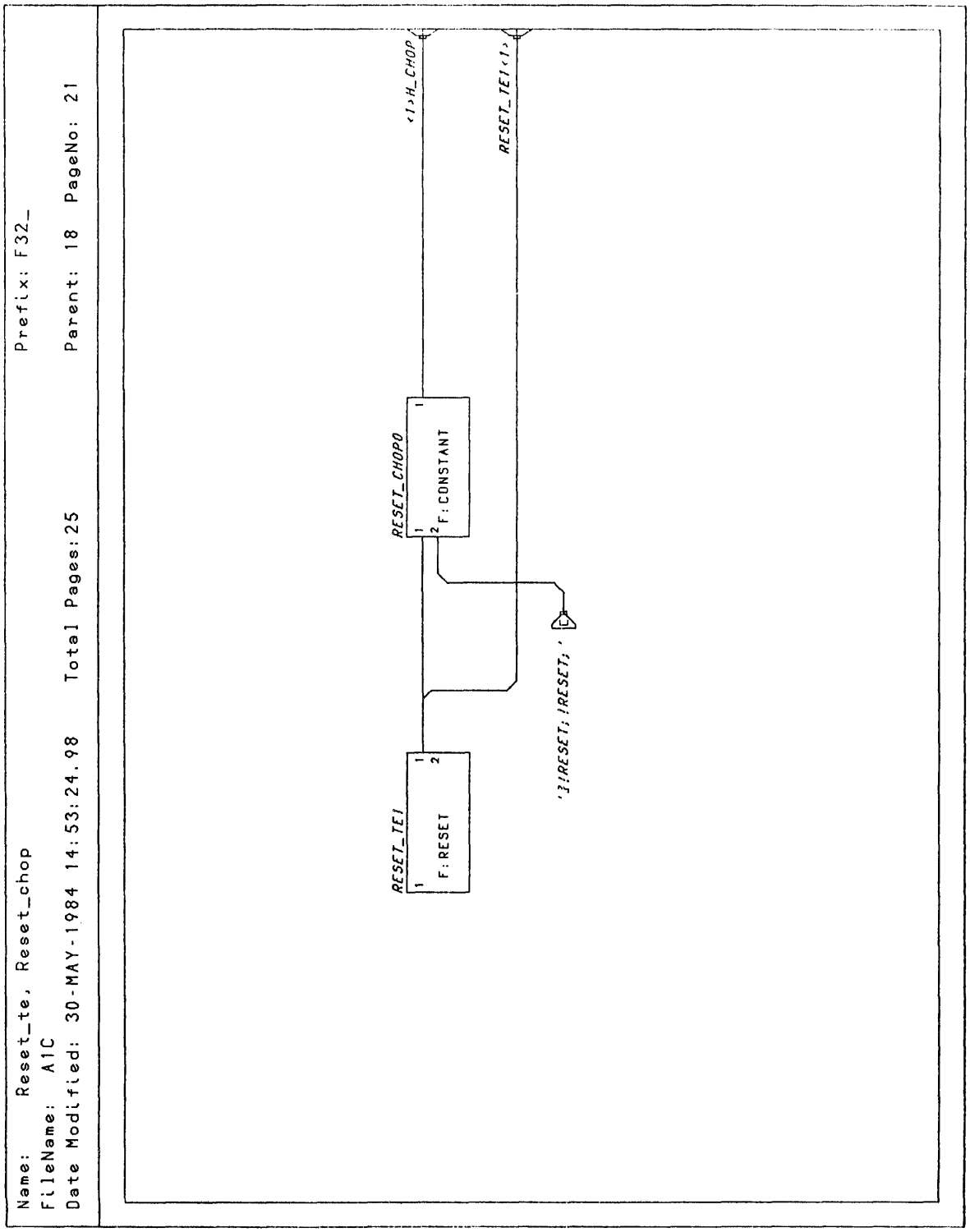


Figure 8-22. PS 390 Host Input Data Flow (RS-232 Interface)

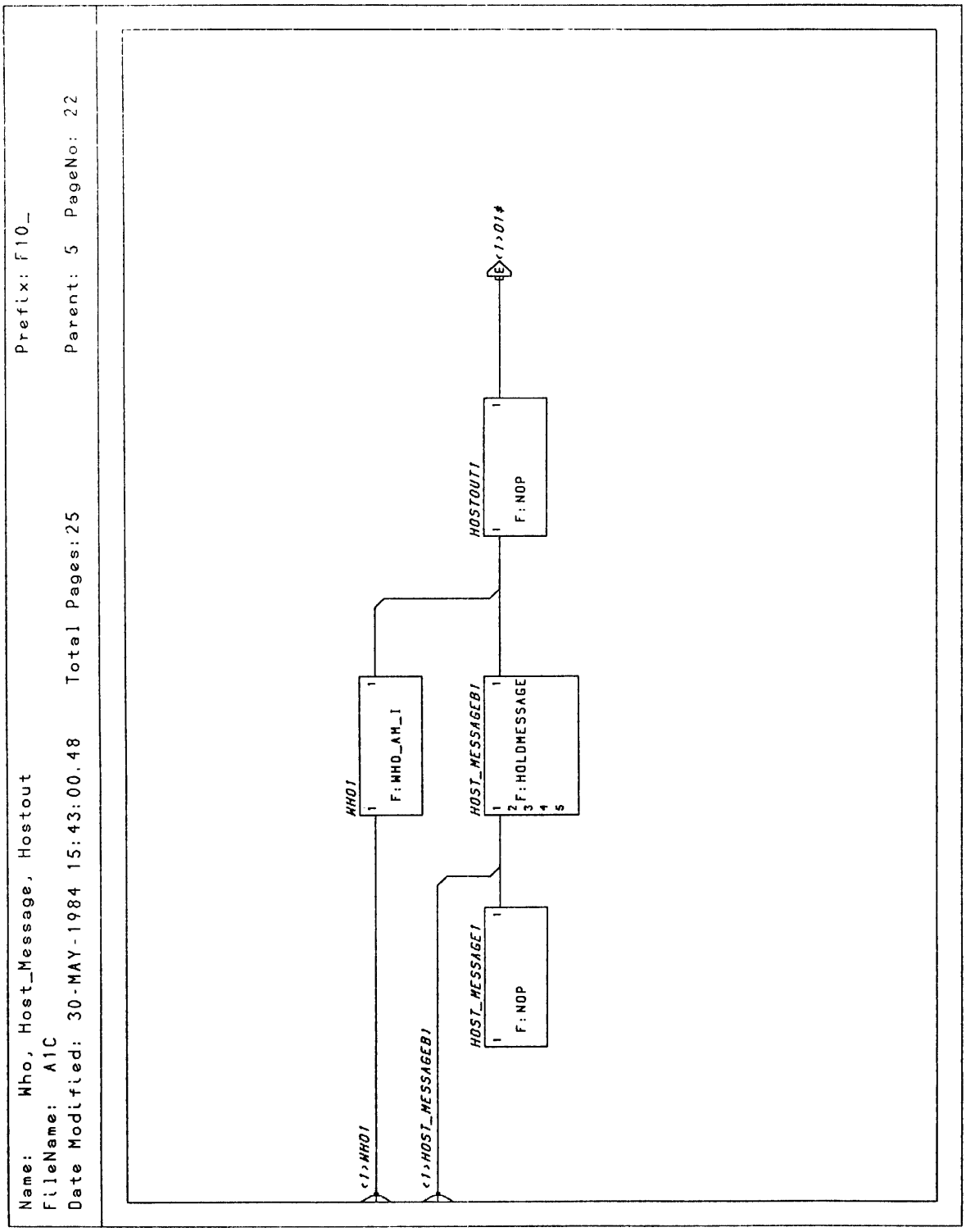


Figure 8-23. PS 390 Host Input Data Flow (RS-232 Interface)

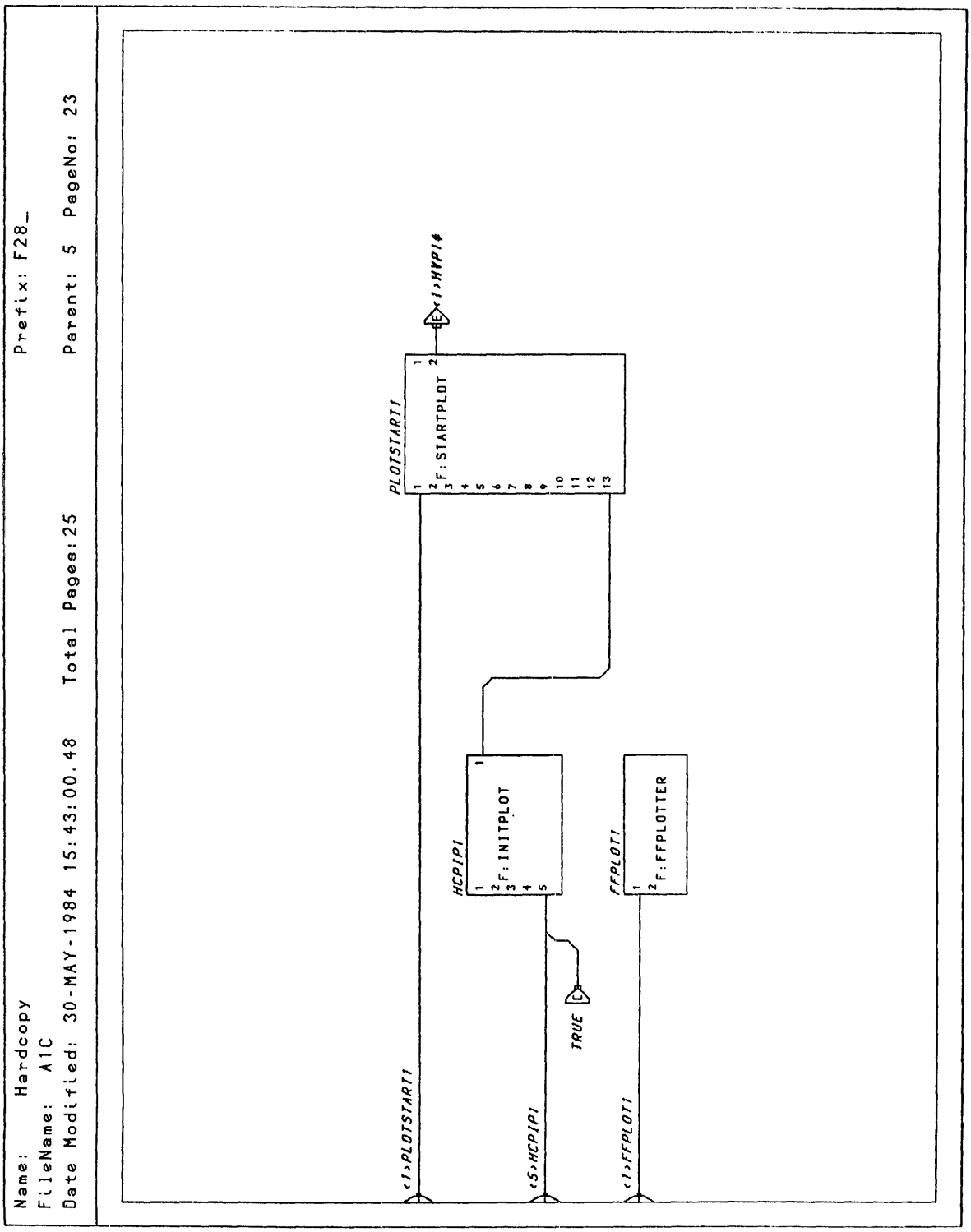


Figure 8-24. PS 390 Host Input Data Flow (RS-232 Interface)

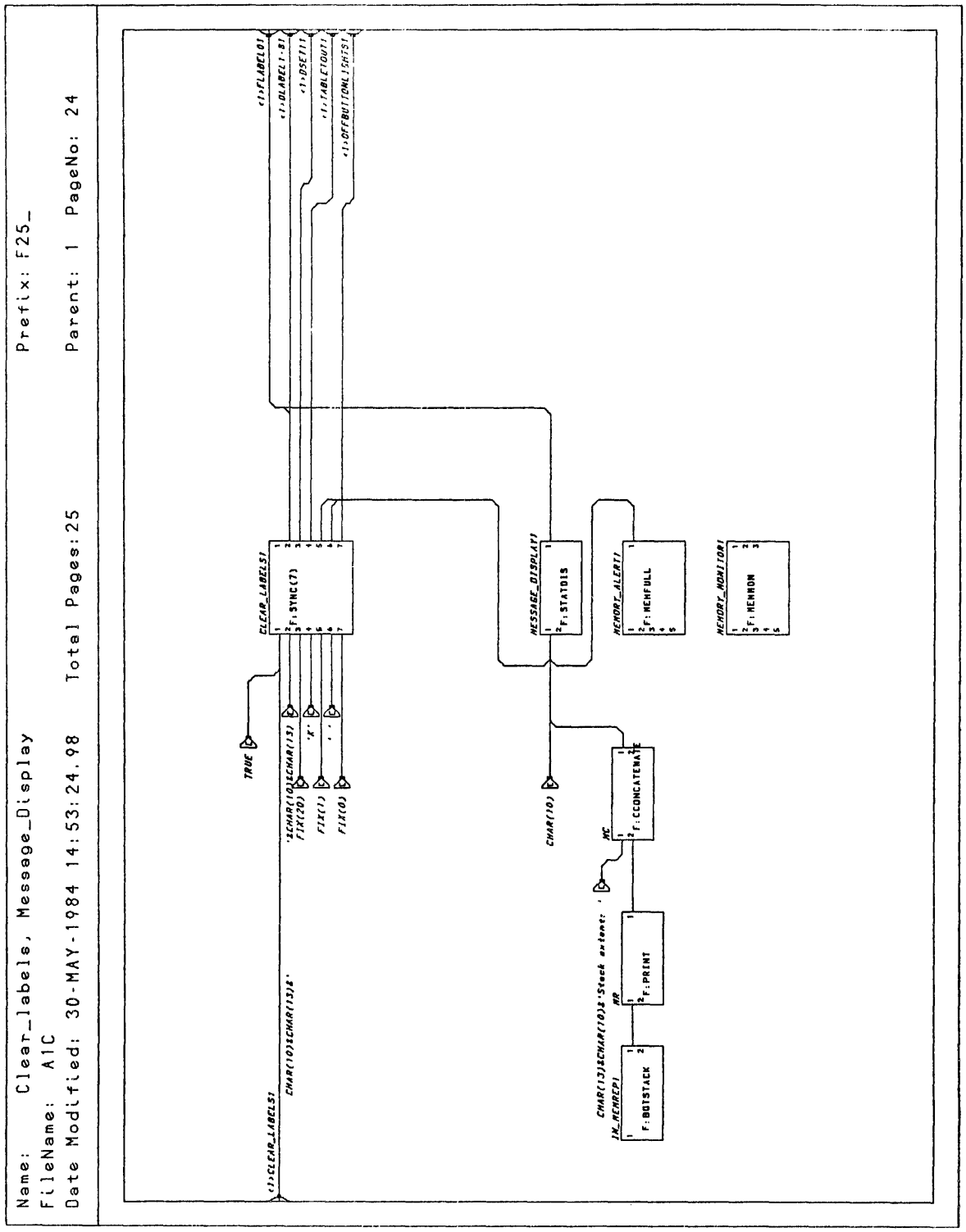


Figure 8-25. PS 390 Host Input Data Flow (RS-232 Interface)

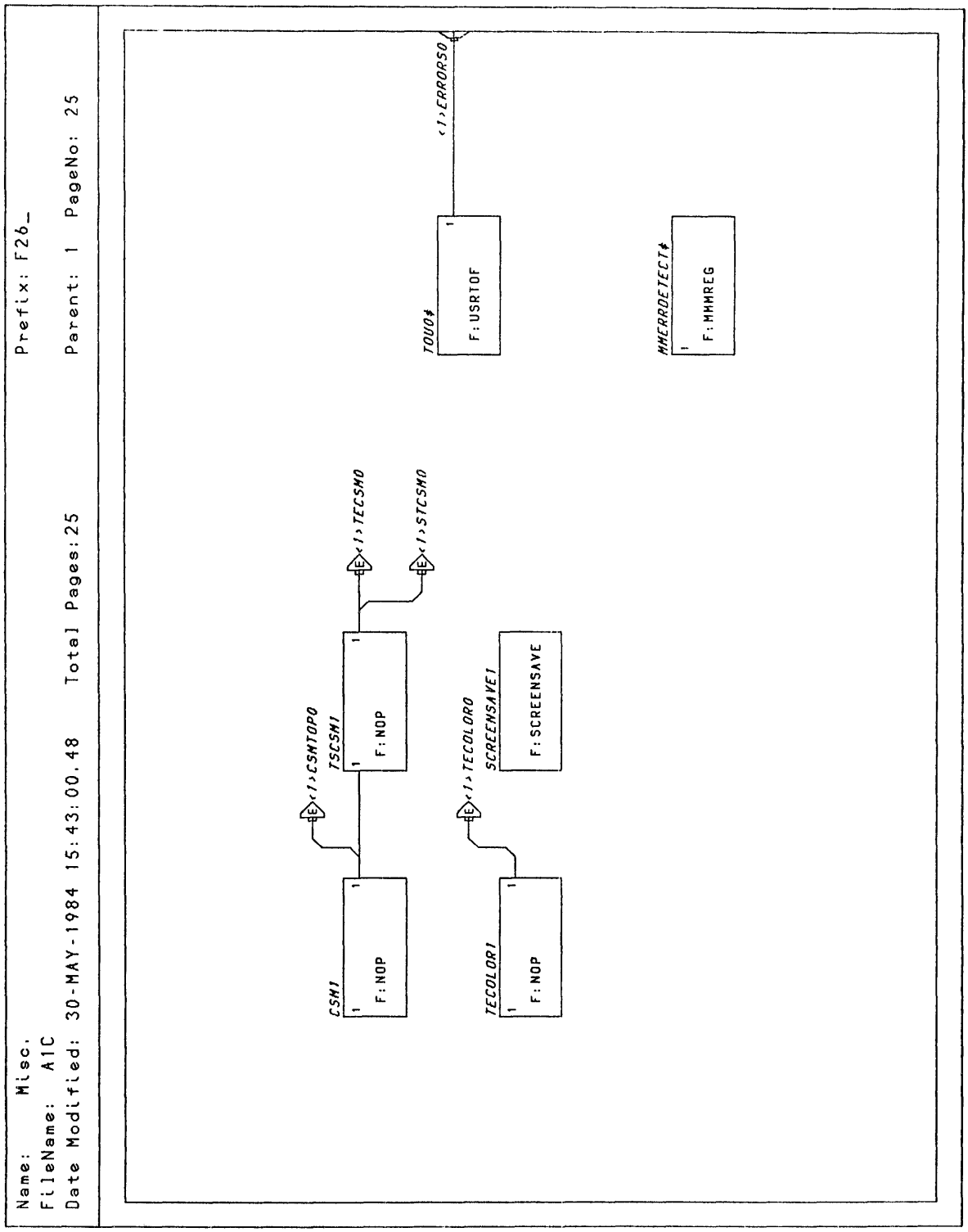


Figure 8-26. PS 390 Host Input Data Flow (RS-232 Interface)

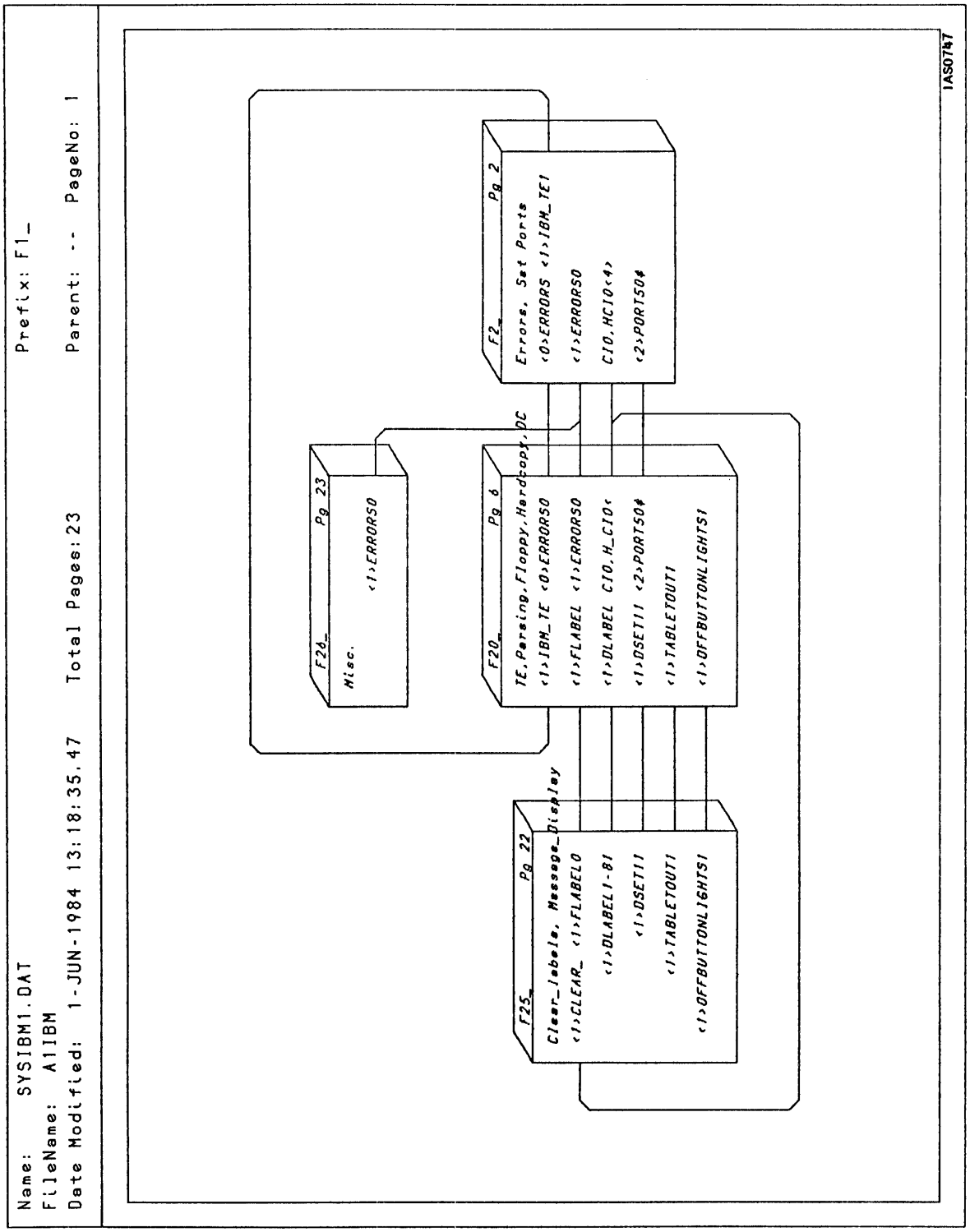


Figure 8-27. PS 390 Host Input Data Flow (IBM)

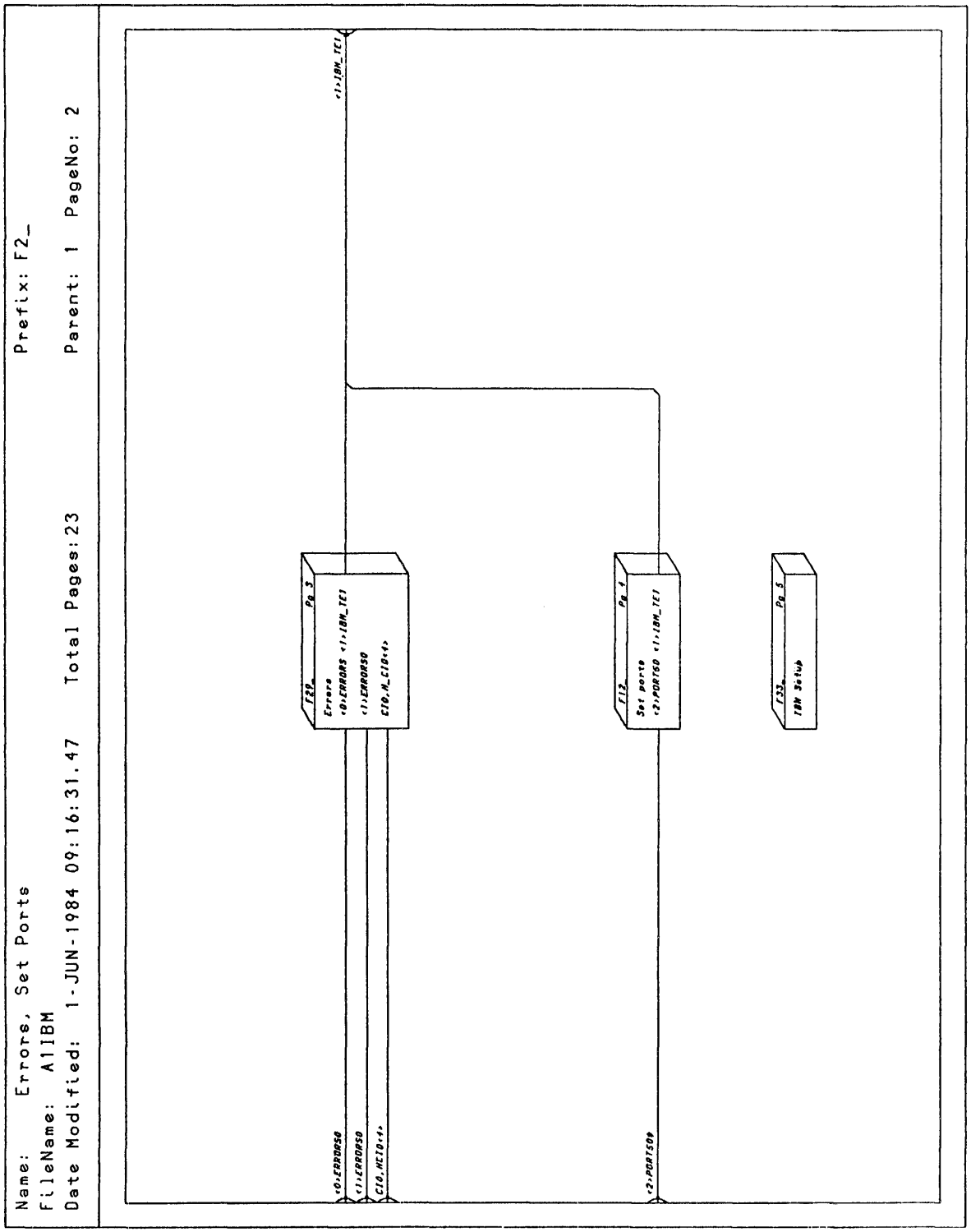


Figure 8-28. PS 390 Host Input Data Flow (IBM)

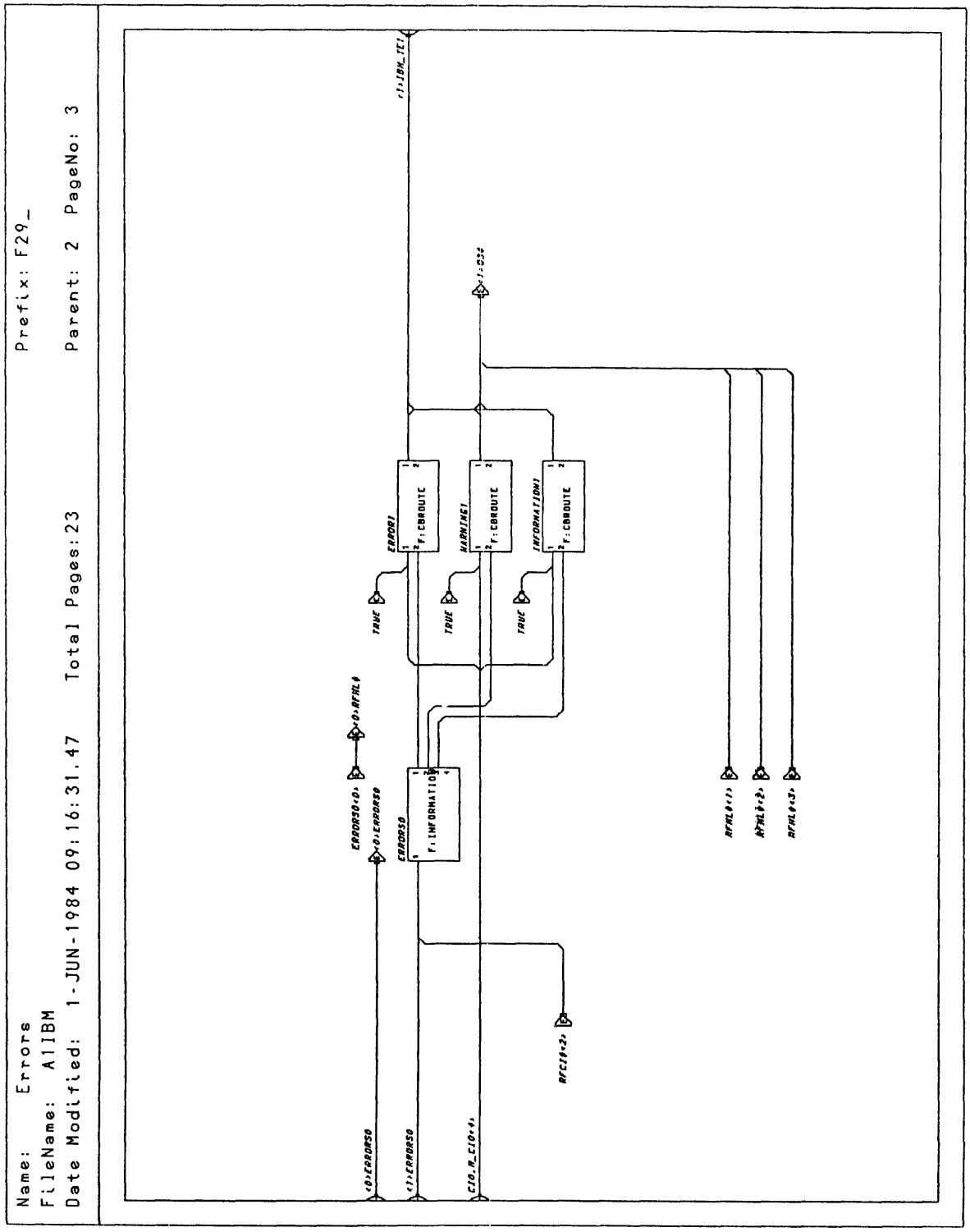


Figure 8-29. PS 390 Host Input Data Flow (IBM)

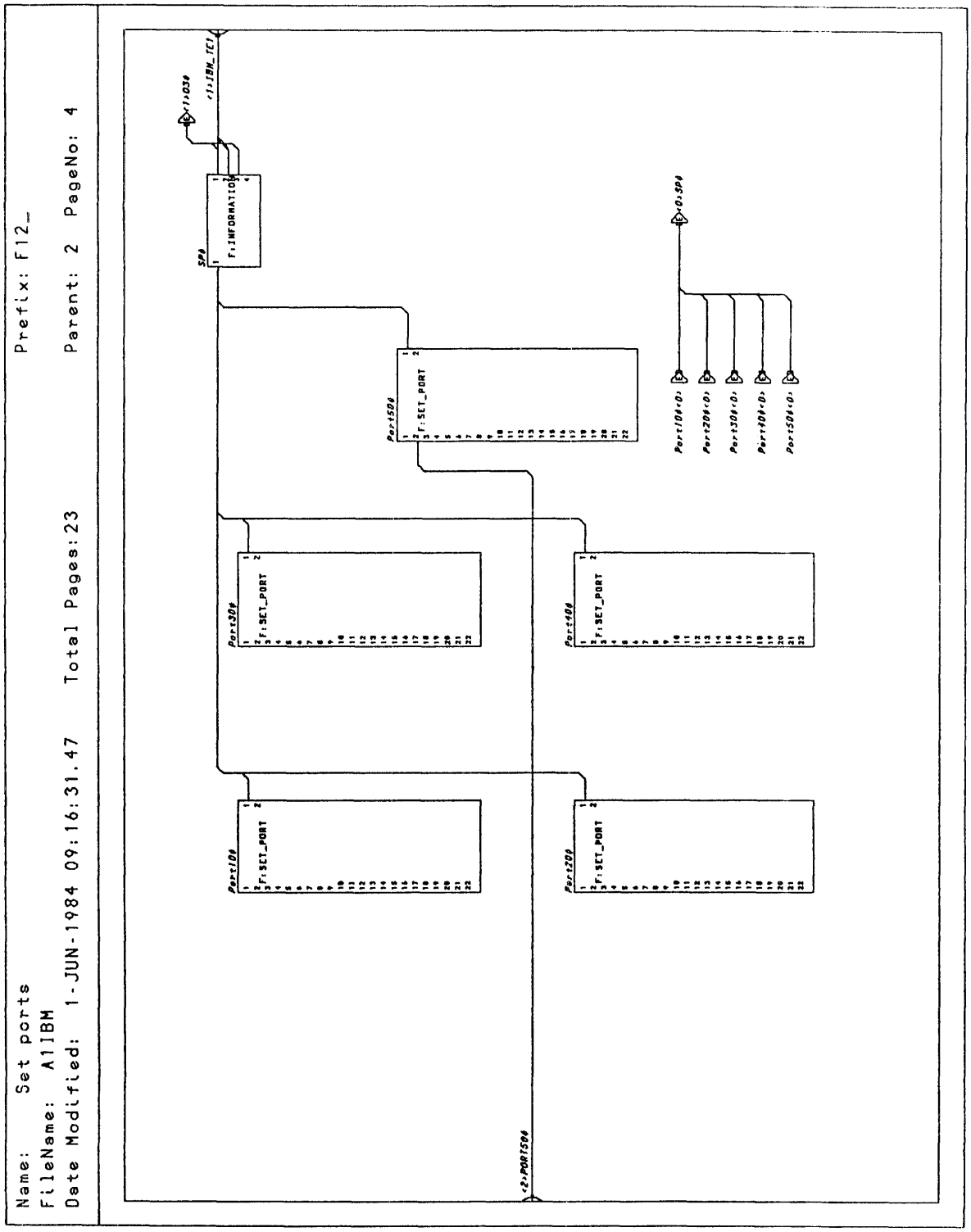


Figure 8-30. PS 390 Host Input Data Flow (IBM)

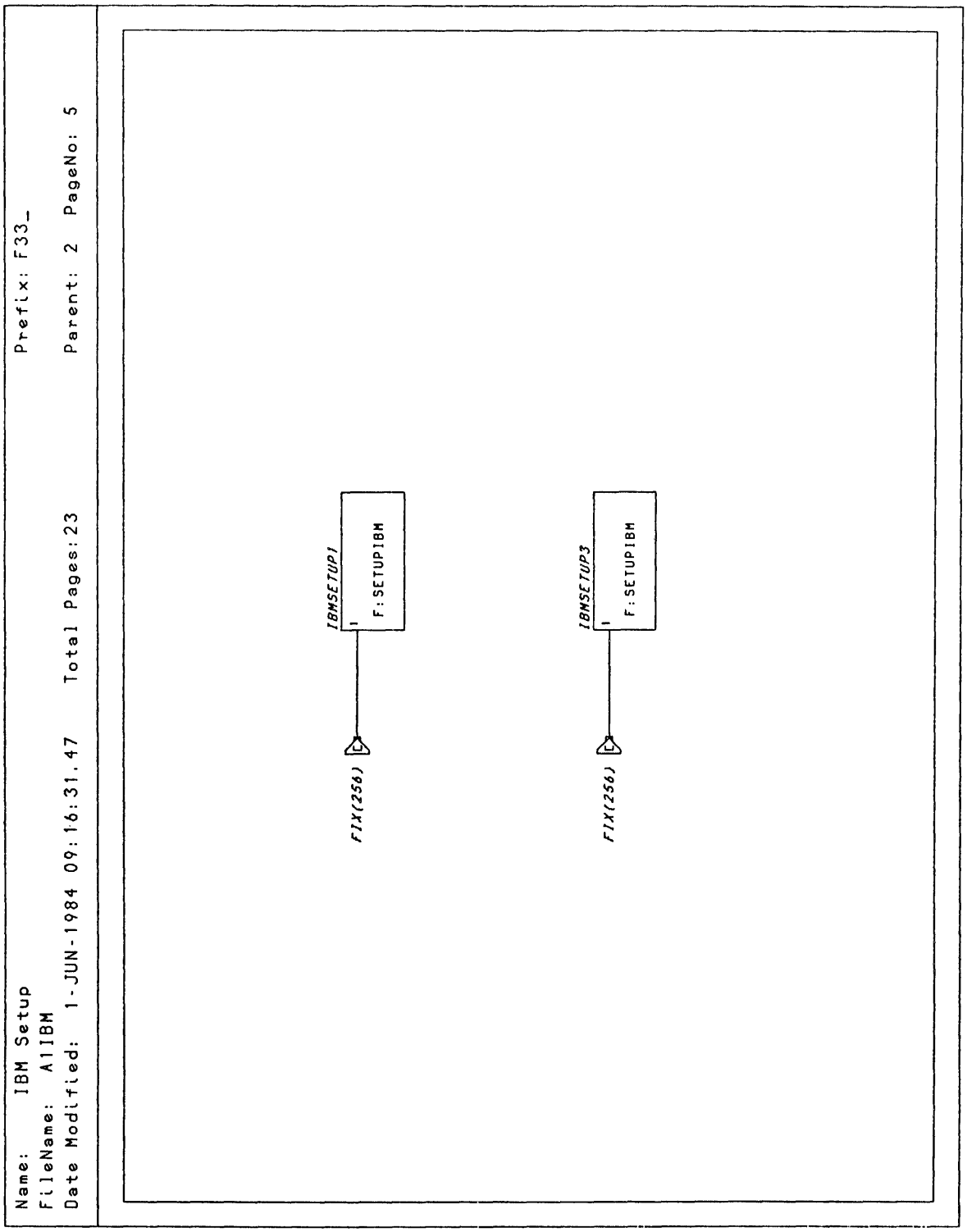


Figure 8-31. PS 390 Host Input Data Flow (IBM)

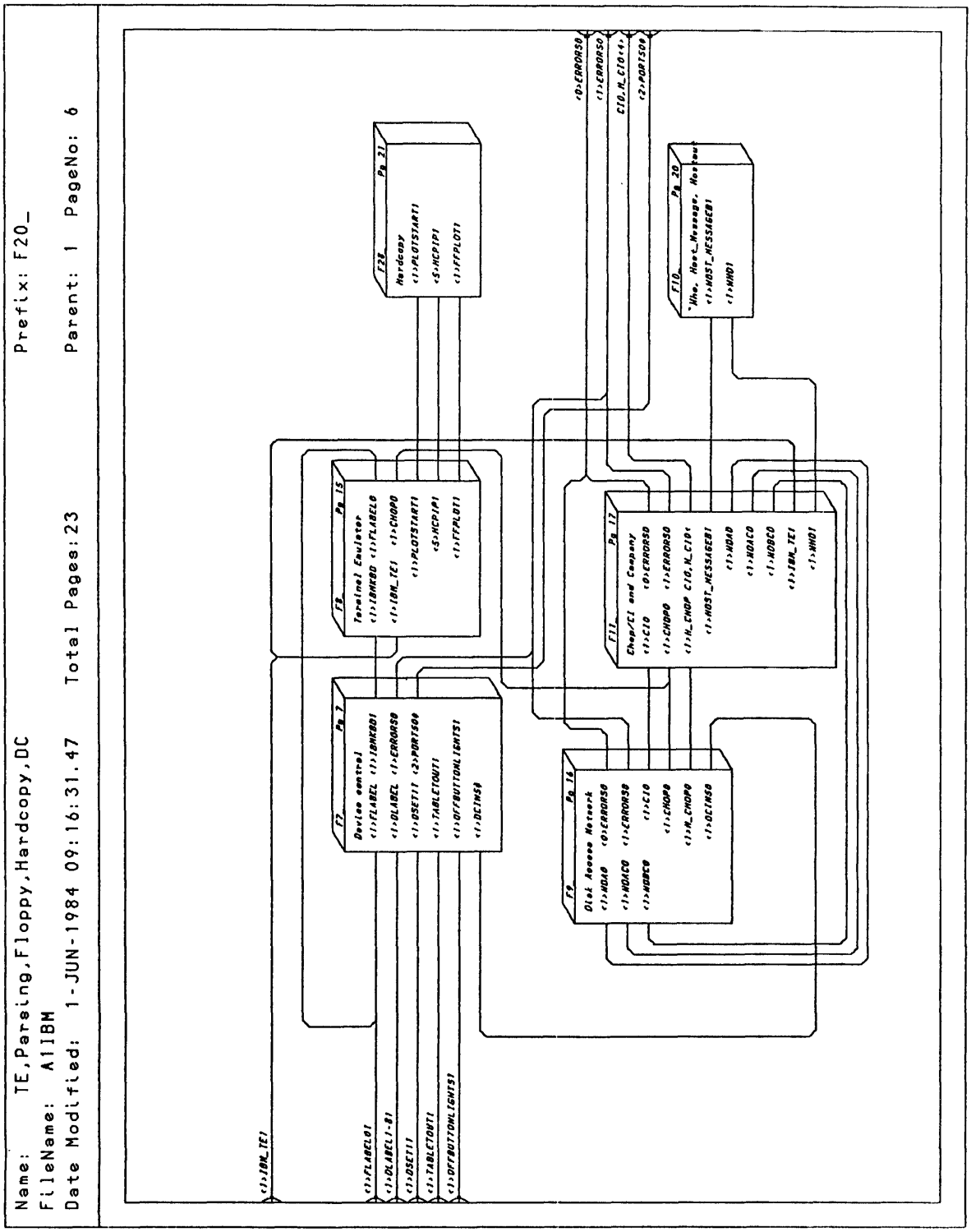


Figure 8-32. PS 390 Host Input Data Flow (IBM)

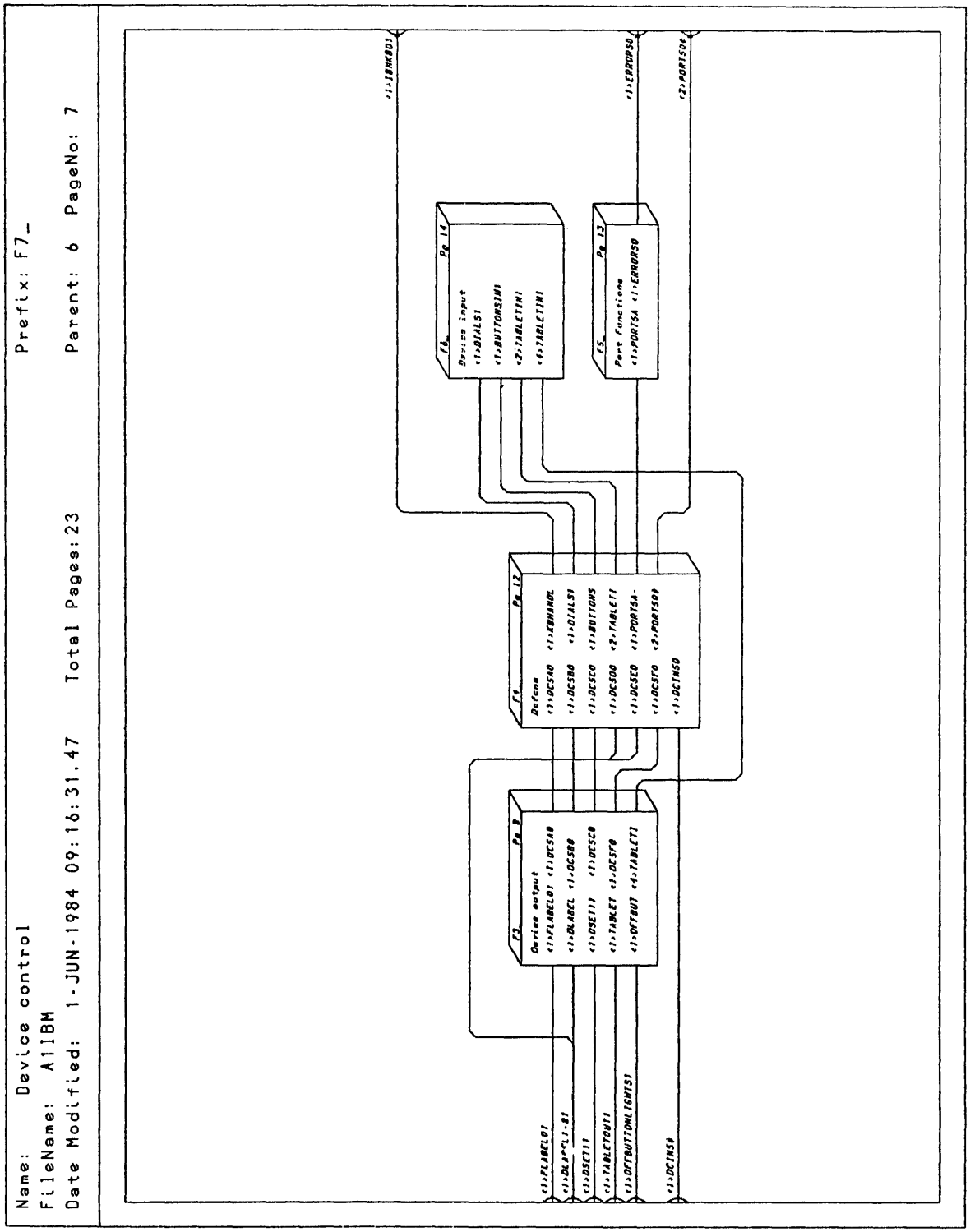


Figure 8-33. PS 390 Host Input Data Flow (IBM)

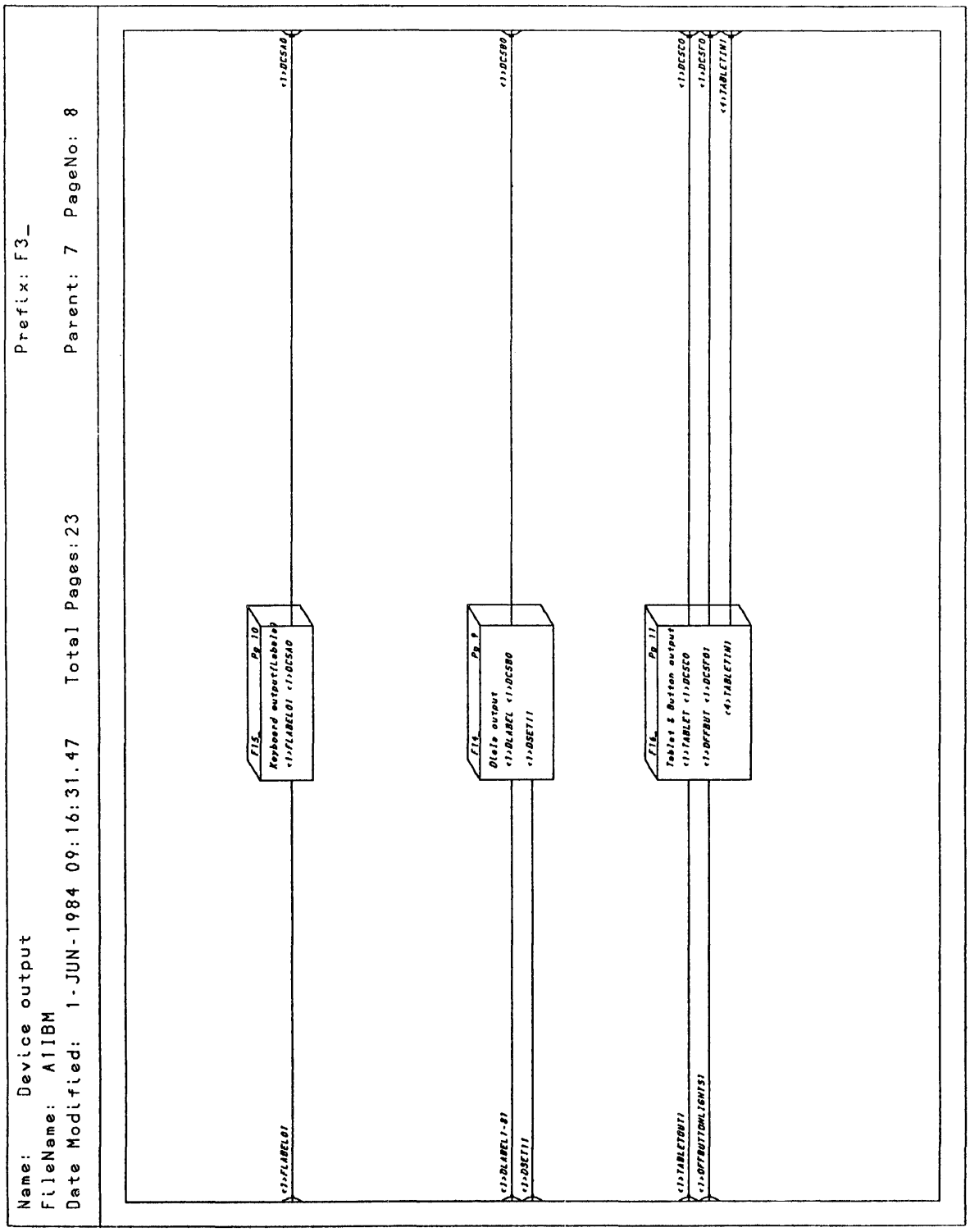


Figure 8-34. PS 390 Host Input Data Flow (IBM)

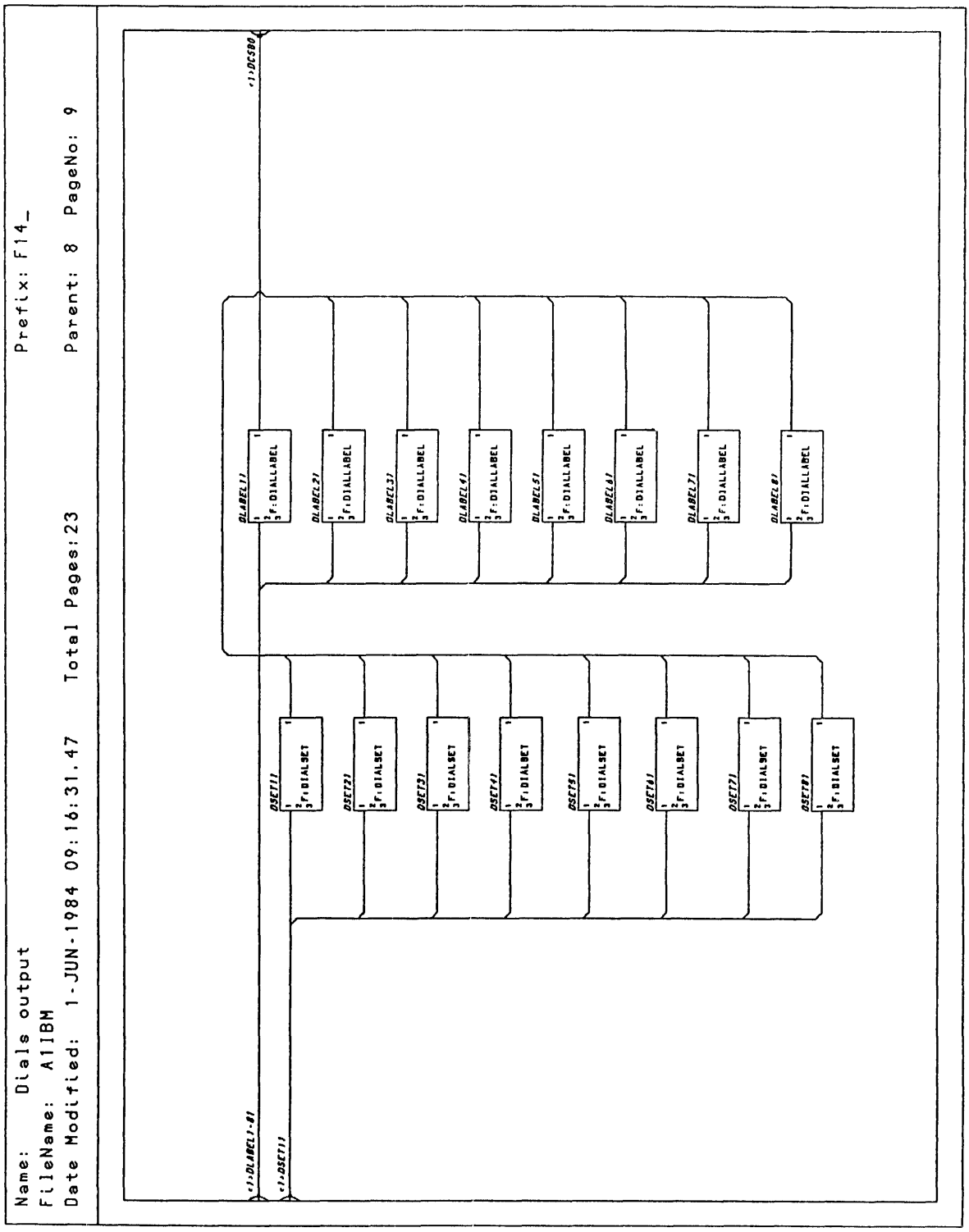


Figure 8-35. PS 390 Host Input Data Flow (IBM)

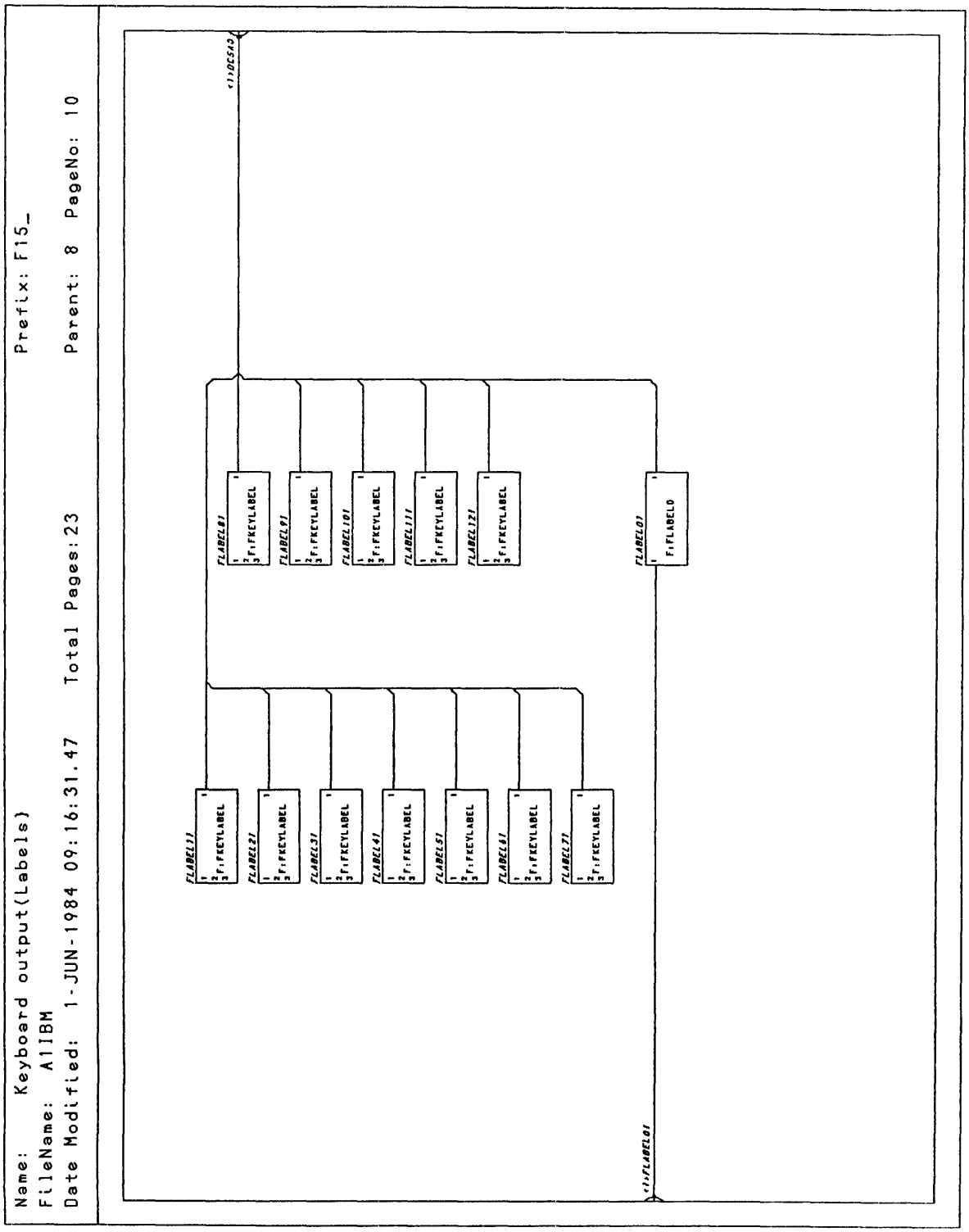


Figure 8-36. PS 390 Host Input Data Flow (IBM)

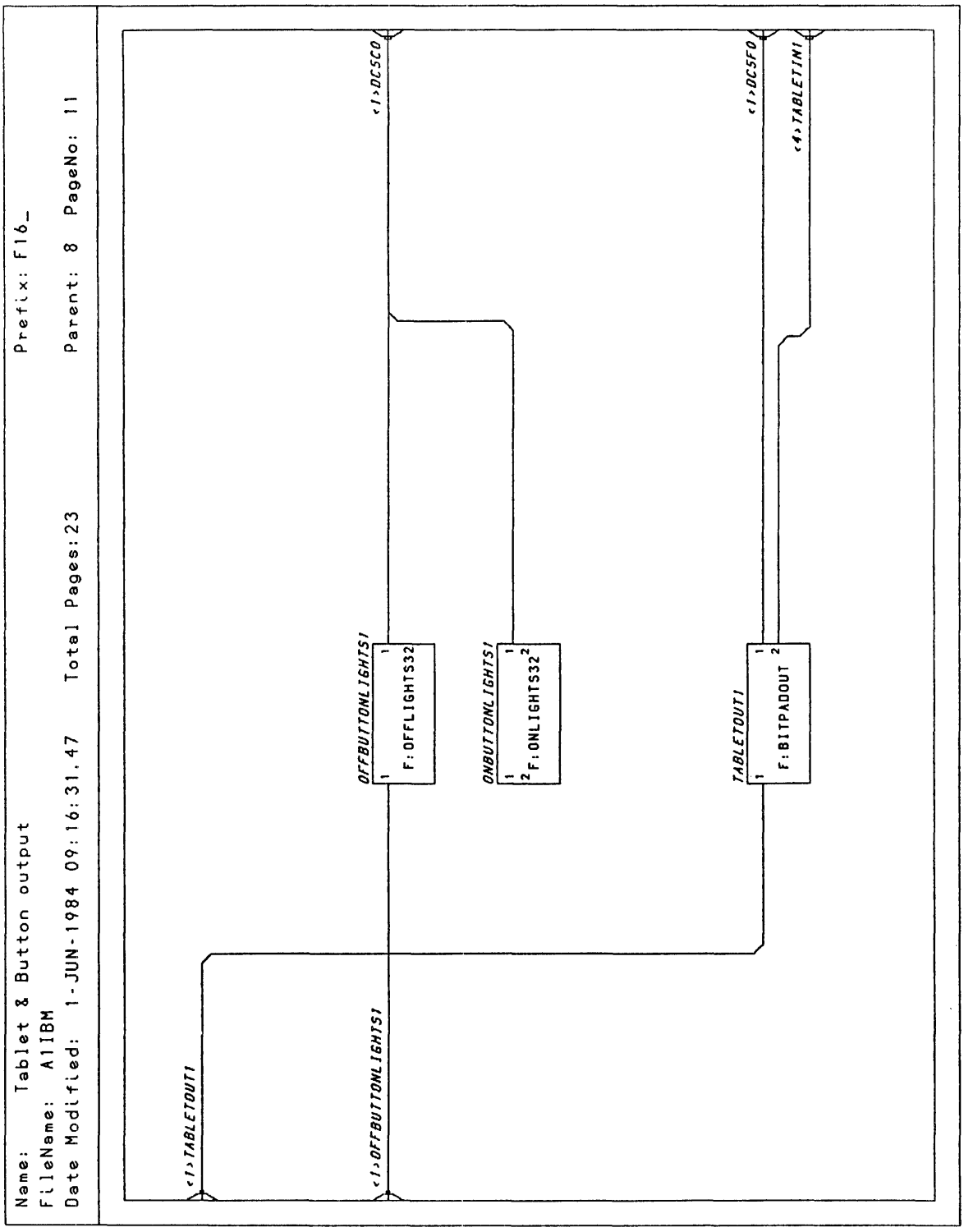


Figure 8-37. PS 390 Host Input Data Flow (IBM)

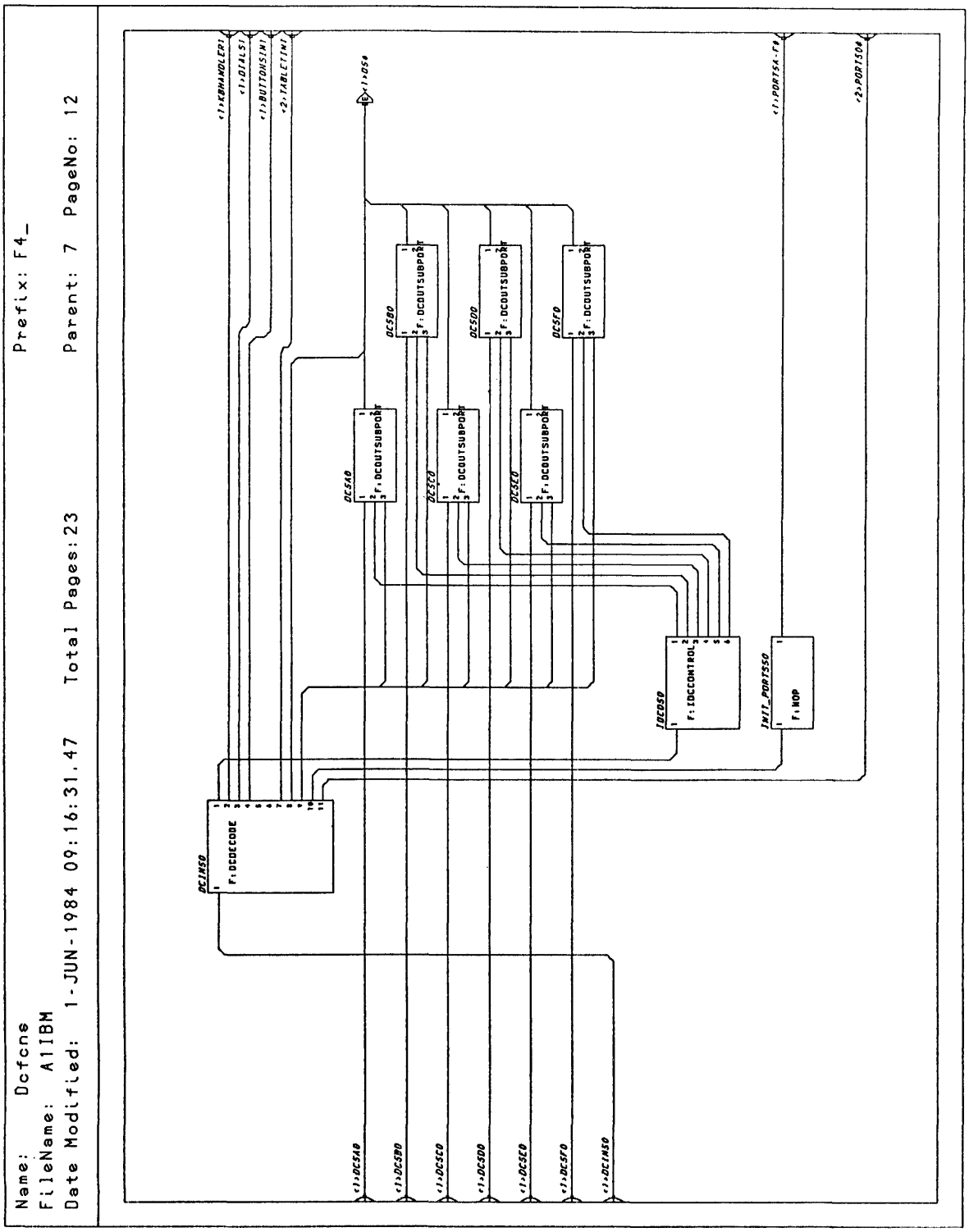


Figure 8-38. PS 390 Host Input Data Flow (IBM)

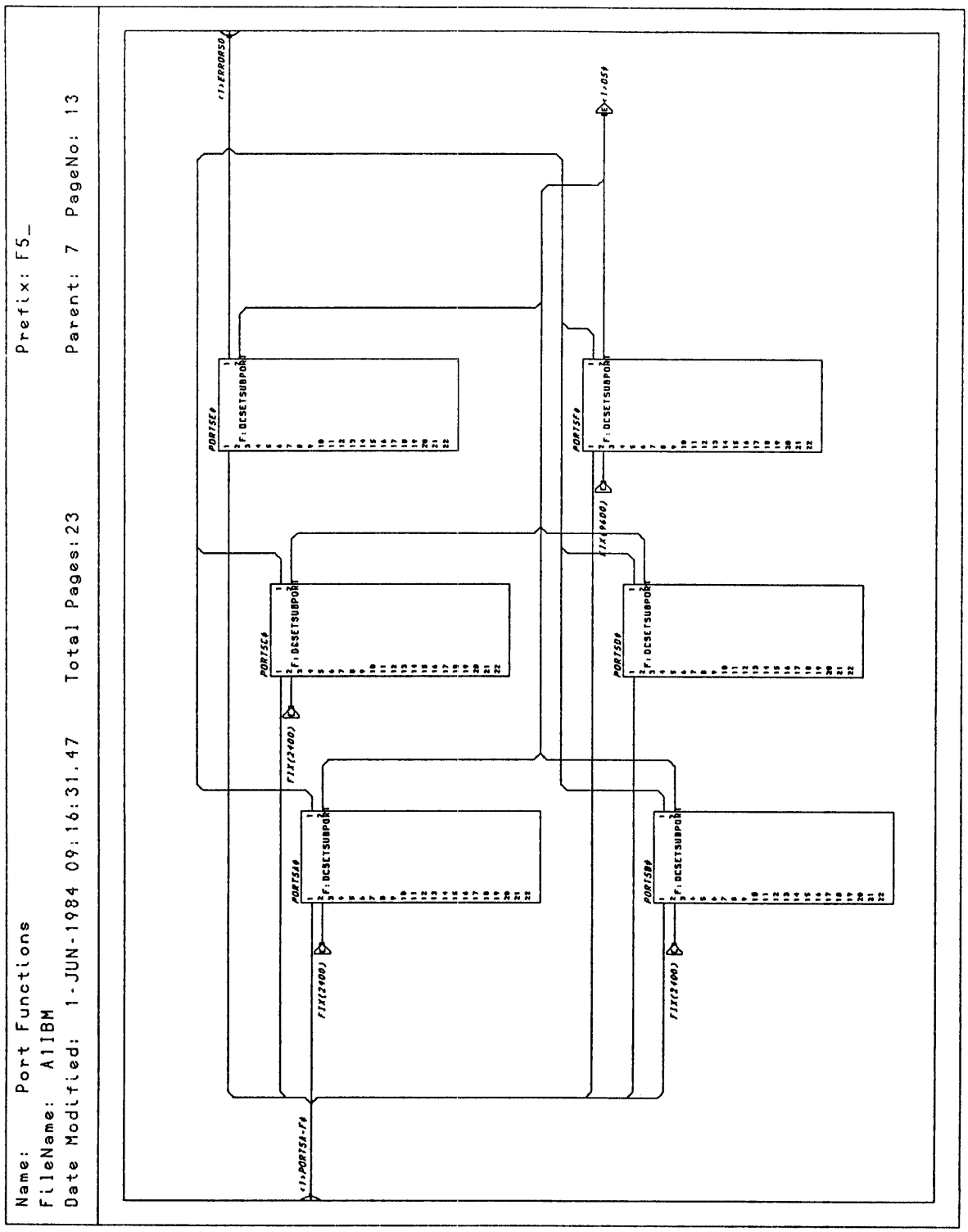


Figure 8-39. PS 390 Host Input Data Flow (IBM)

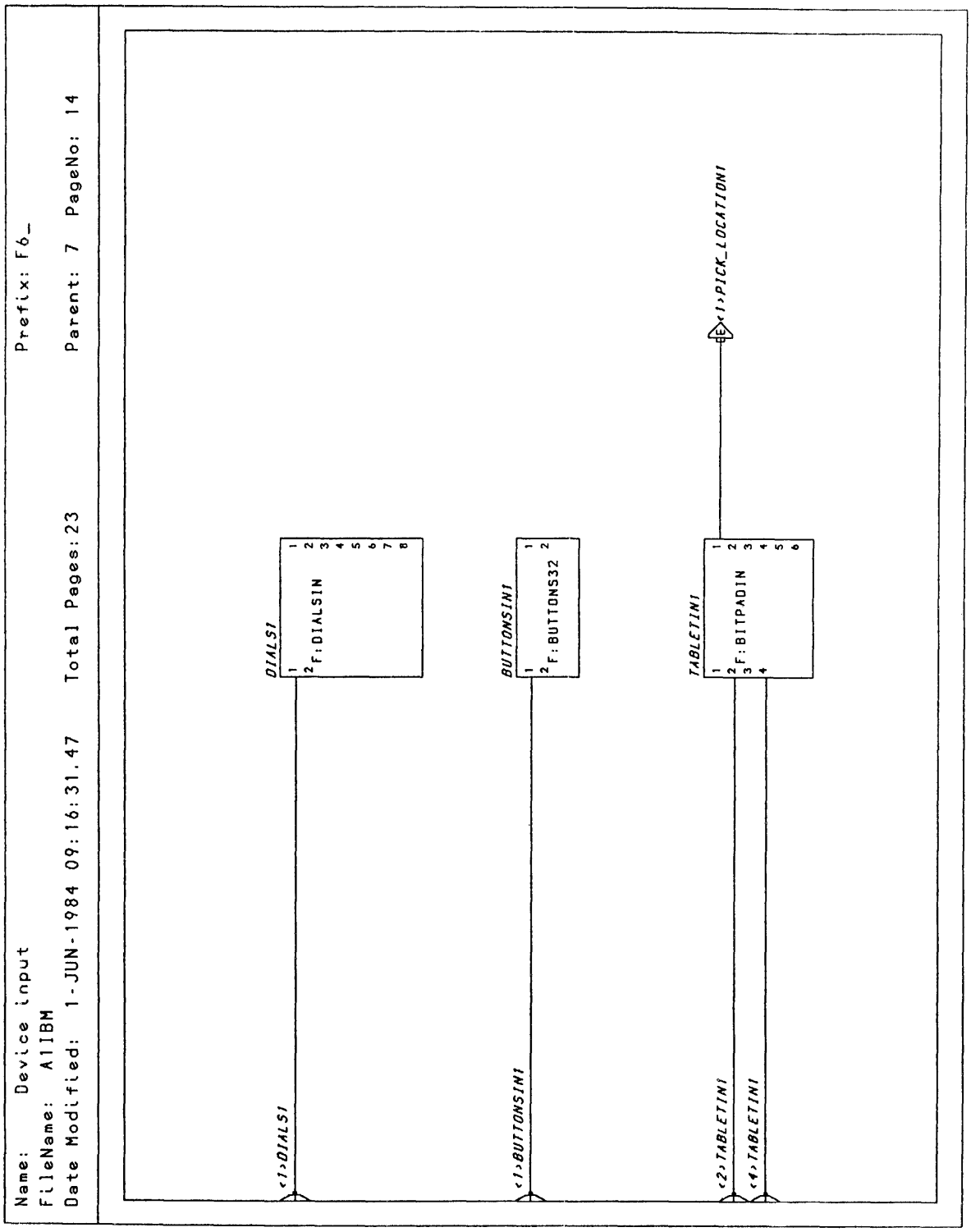


Figure 8-40. PS 390 Host Input Data Flow (IBM)

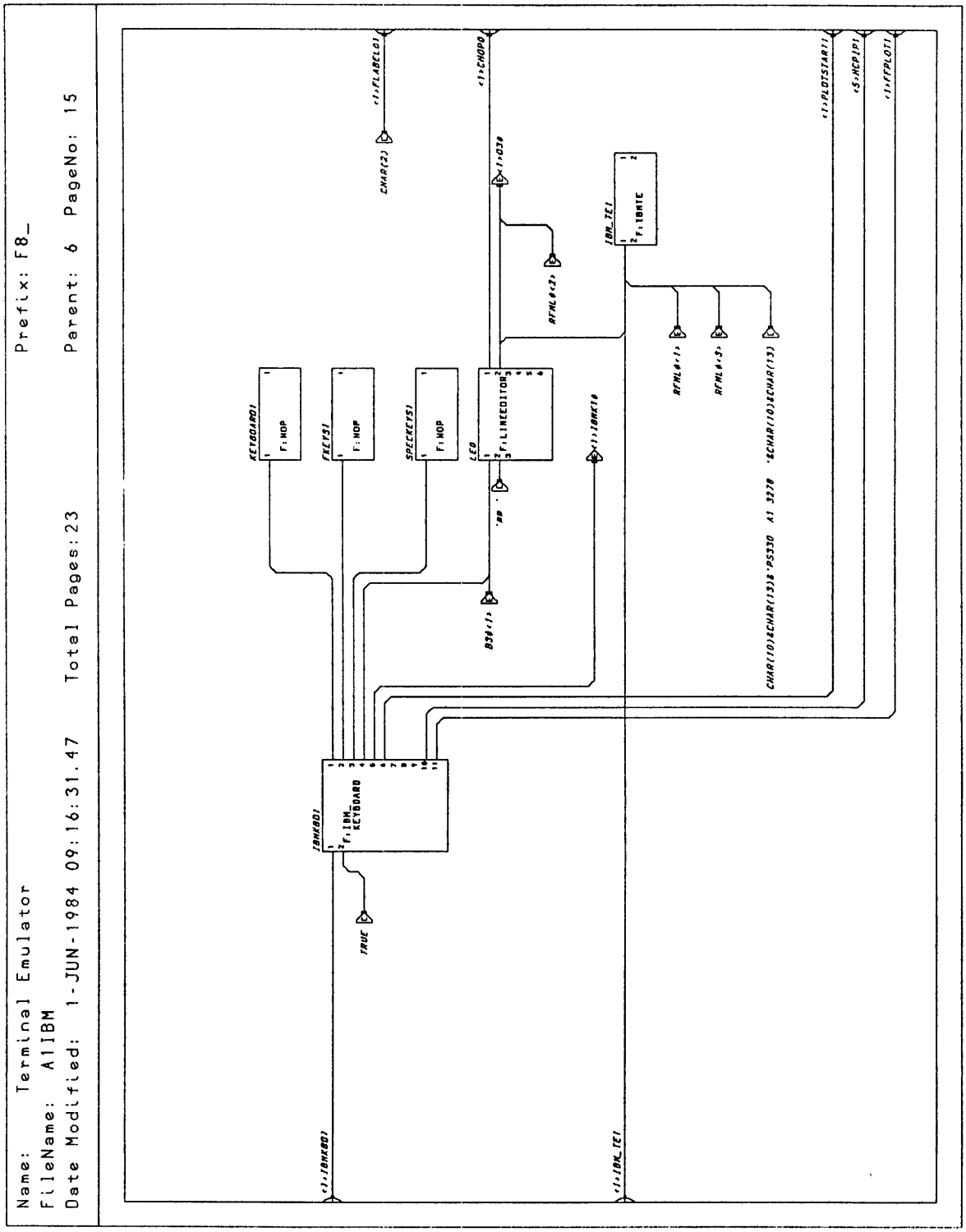


Figure 8-41. PS 390 Host Input Data Flow (IBM)

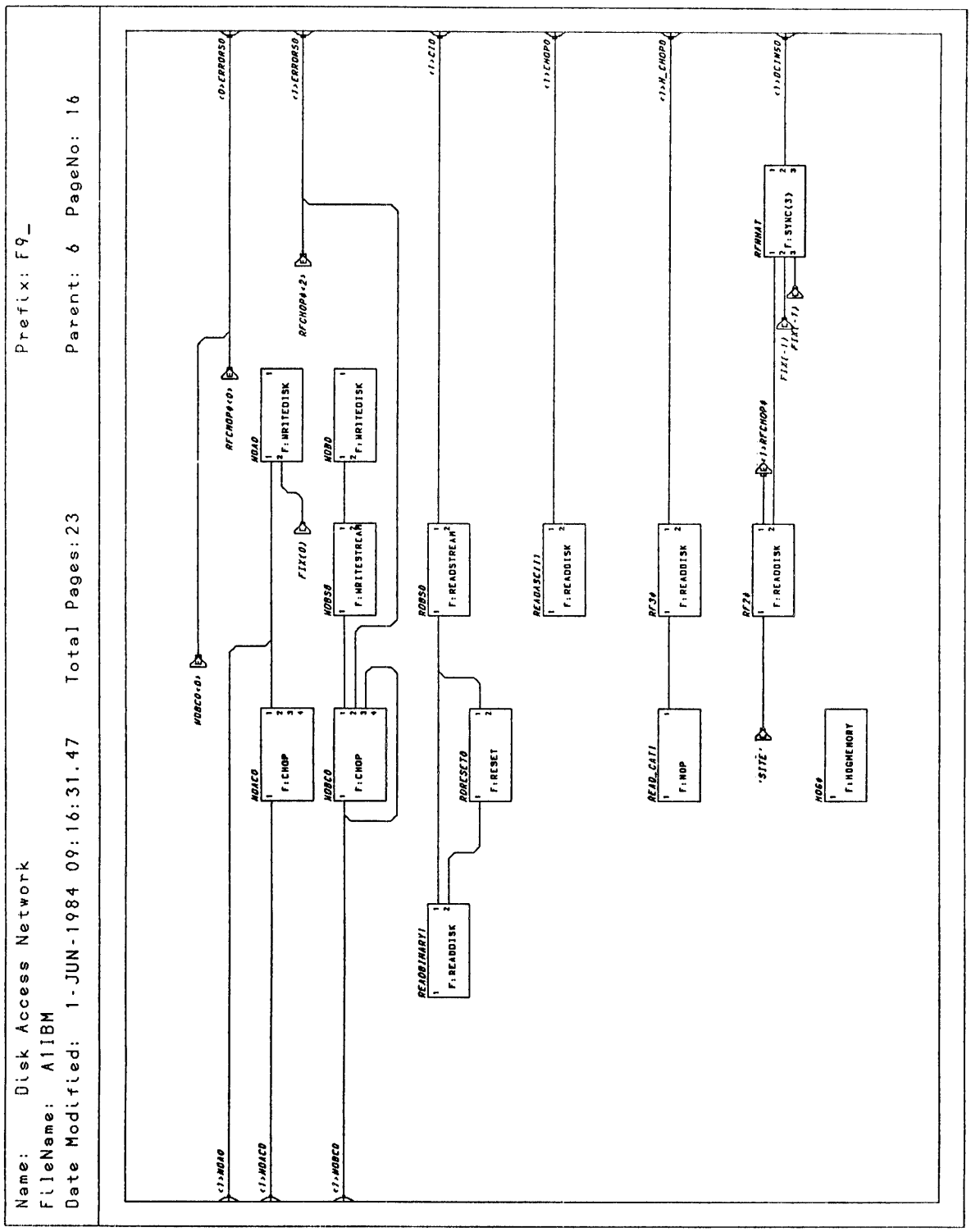


Figure 8-42. PS 390 Host Input Data Flow (IBM)

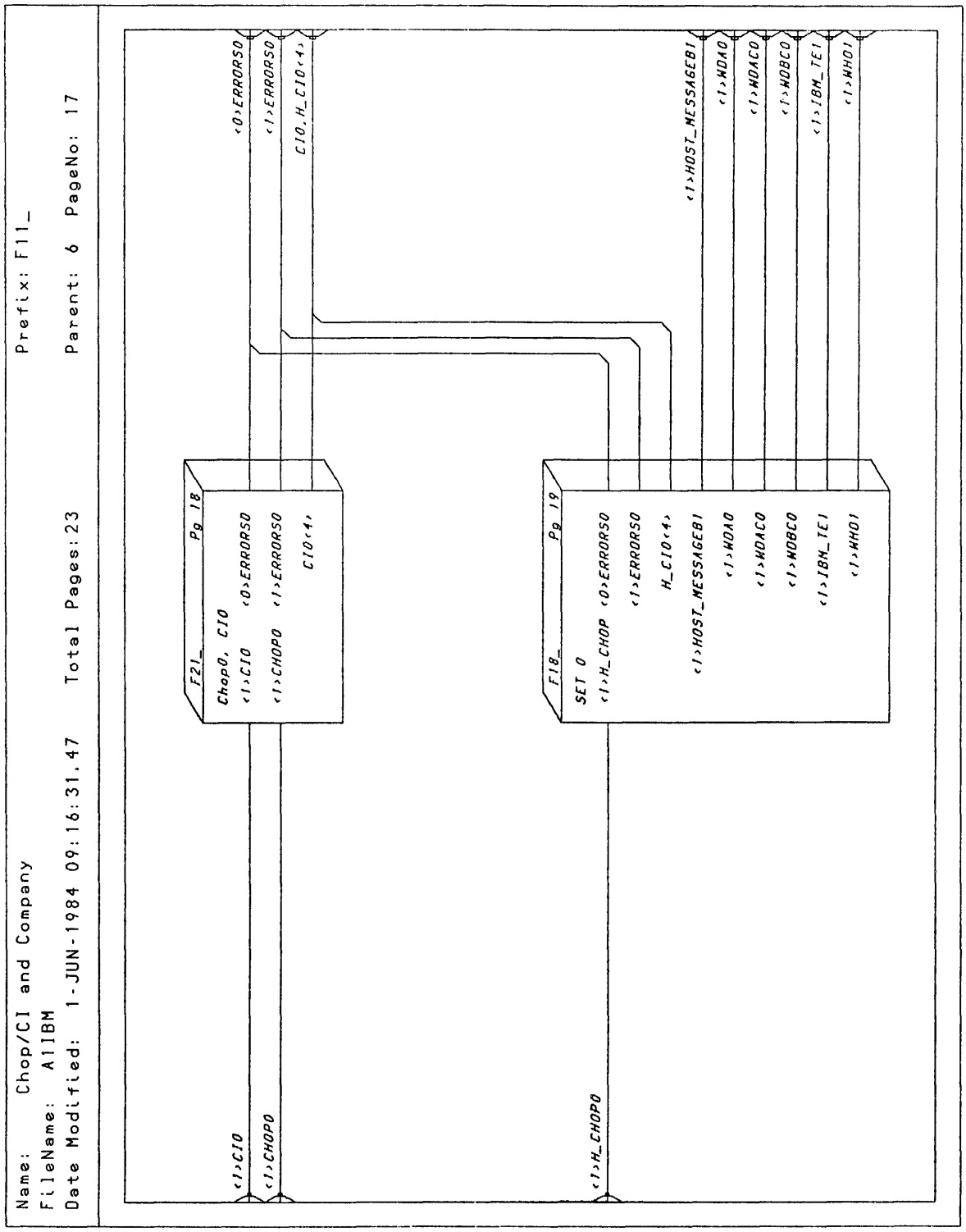


Figure 8-43. PS 390 Host Input Data Flow (IBM)

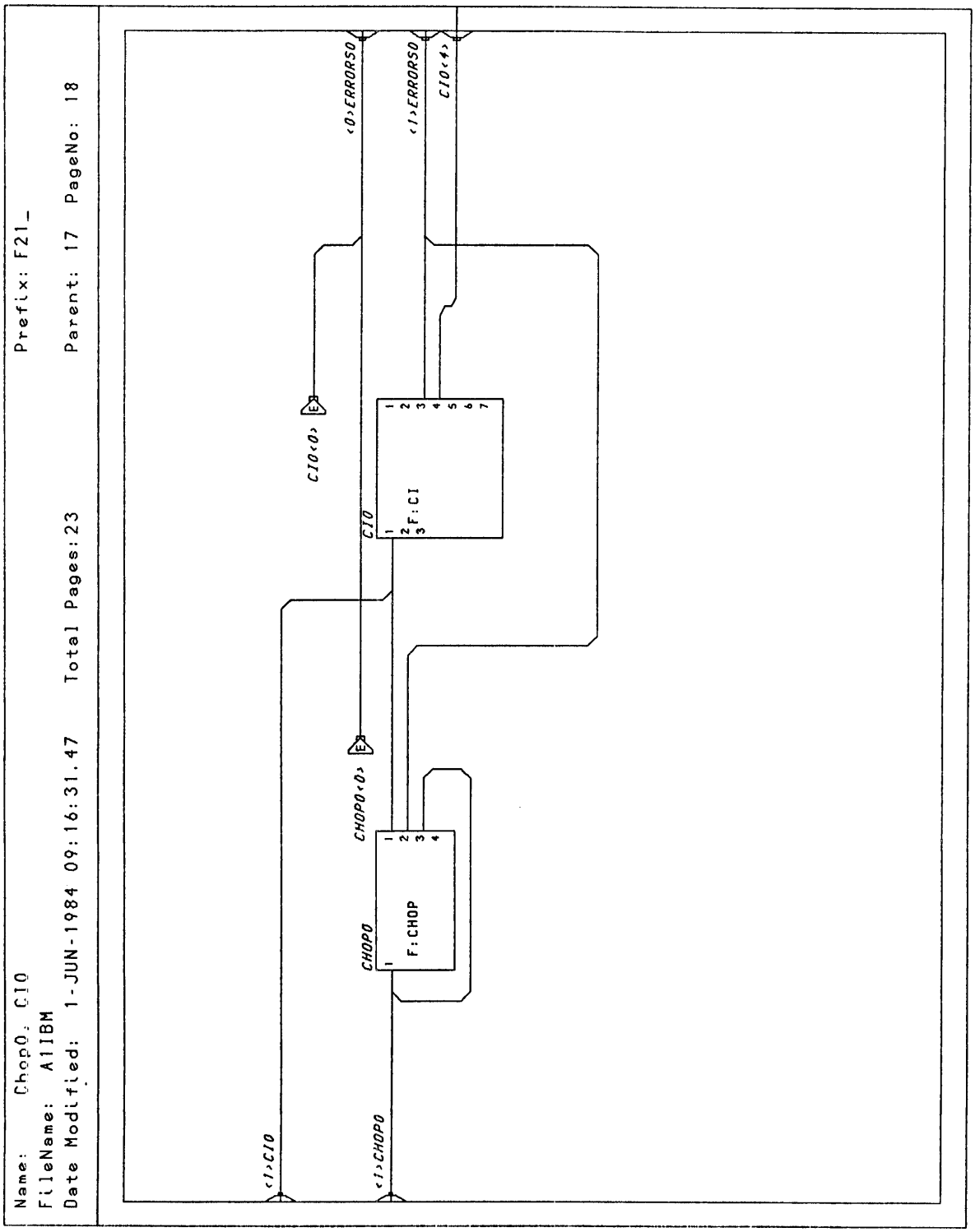


Figure 8-44. PS 390 Host Input Data Flow (IBM)

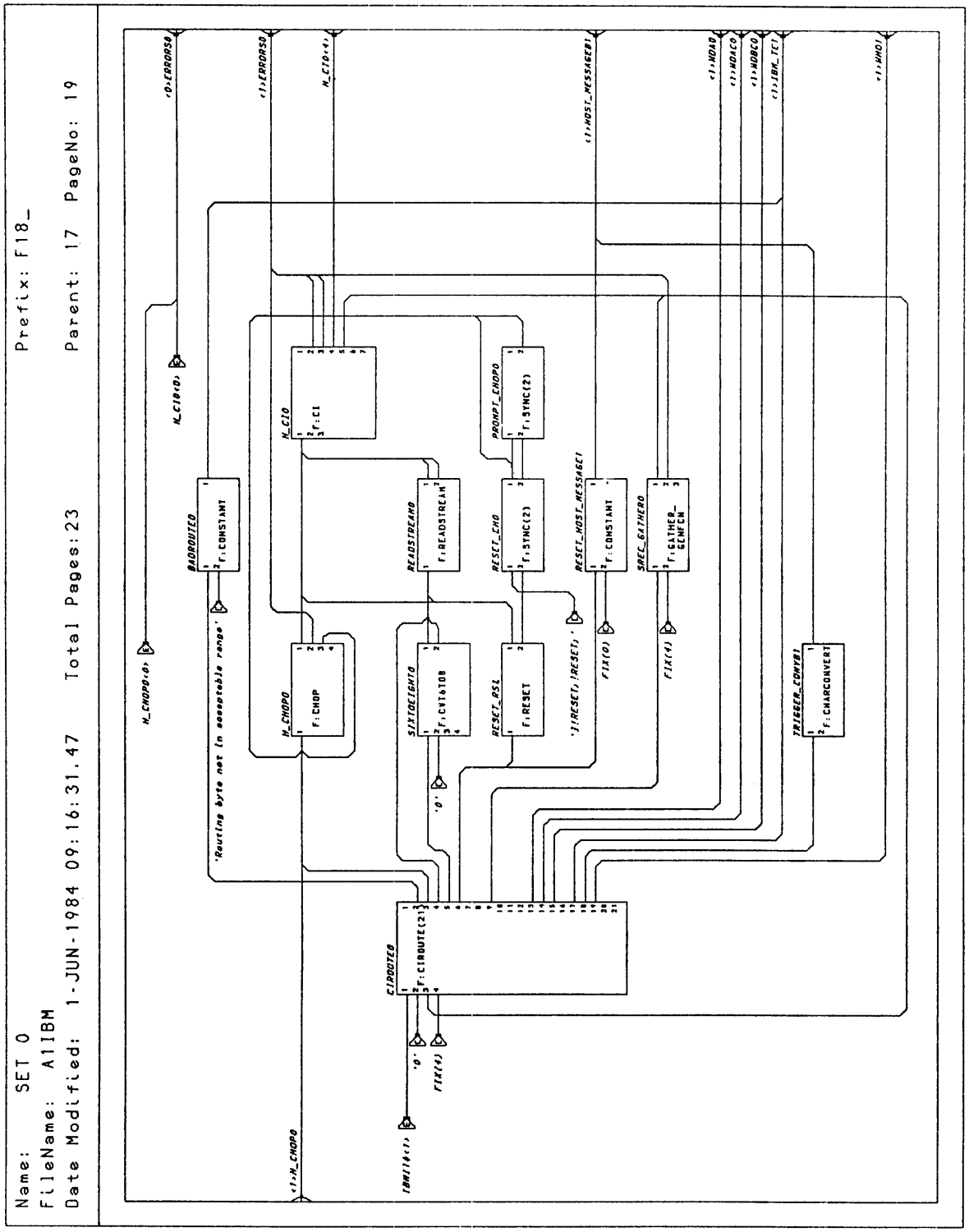


Figure 8-45. PS 390 Host Input Data Flow (IBM)

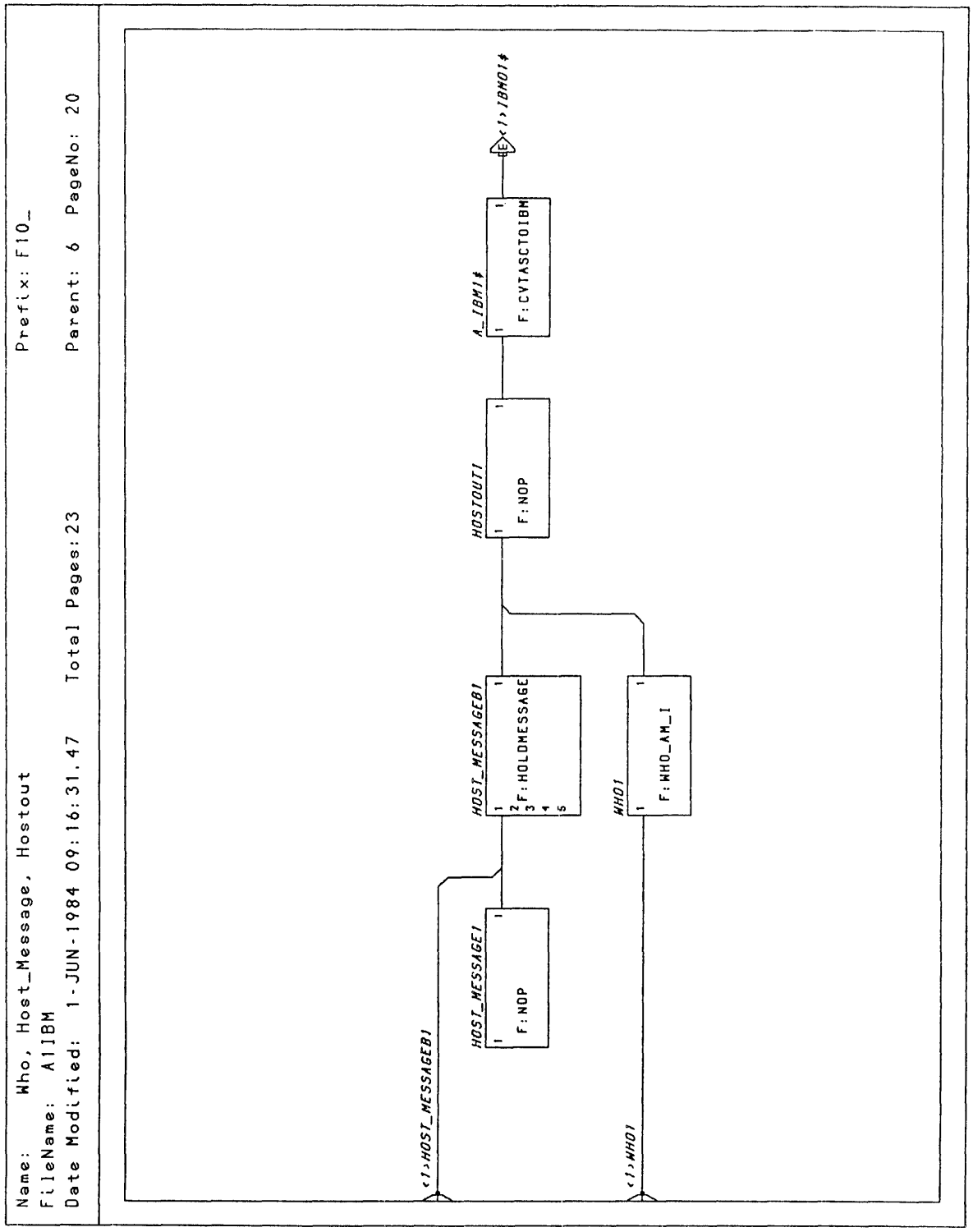


Figure 8-46. PS 390 Host Input Data Flow (IBM)

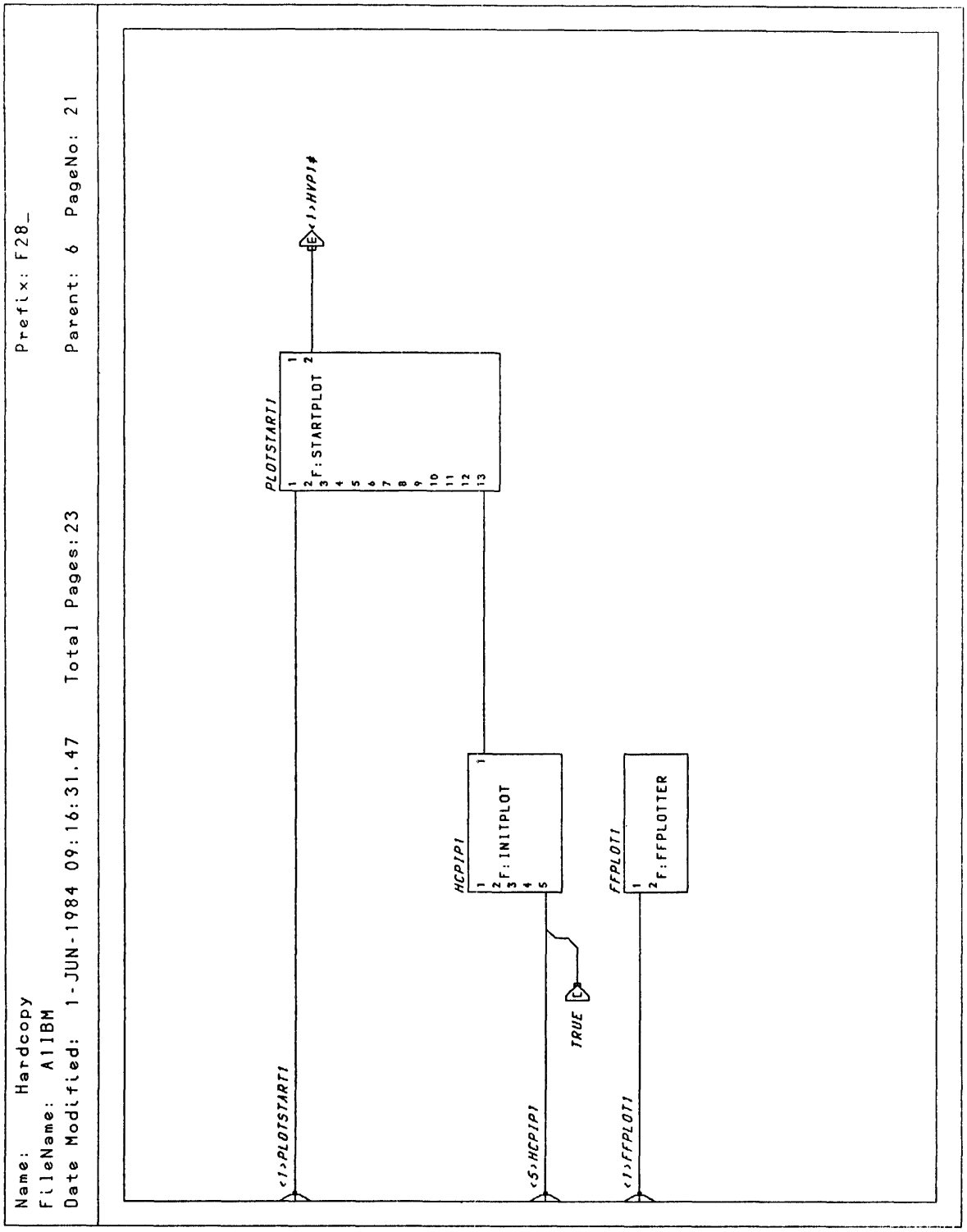


Figure 8-47. PS 390 Host Input Data Flow (IBM)

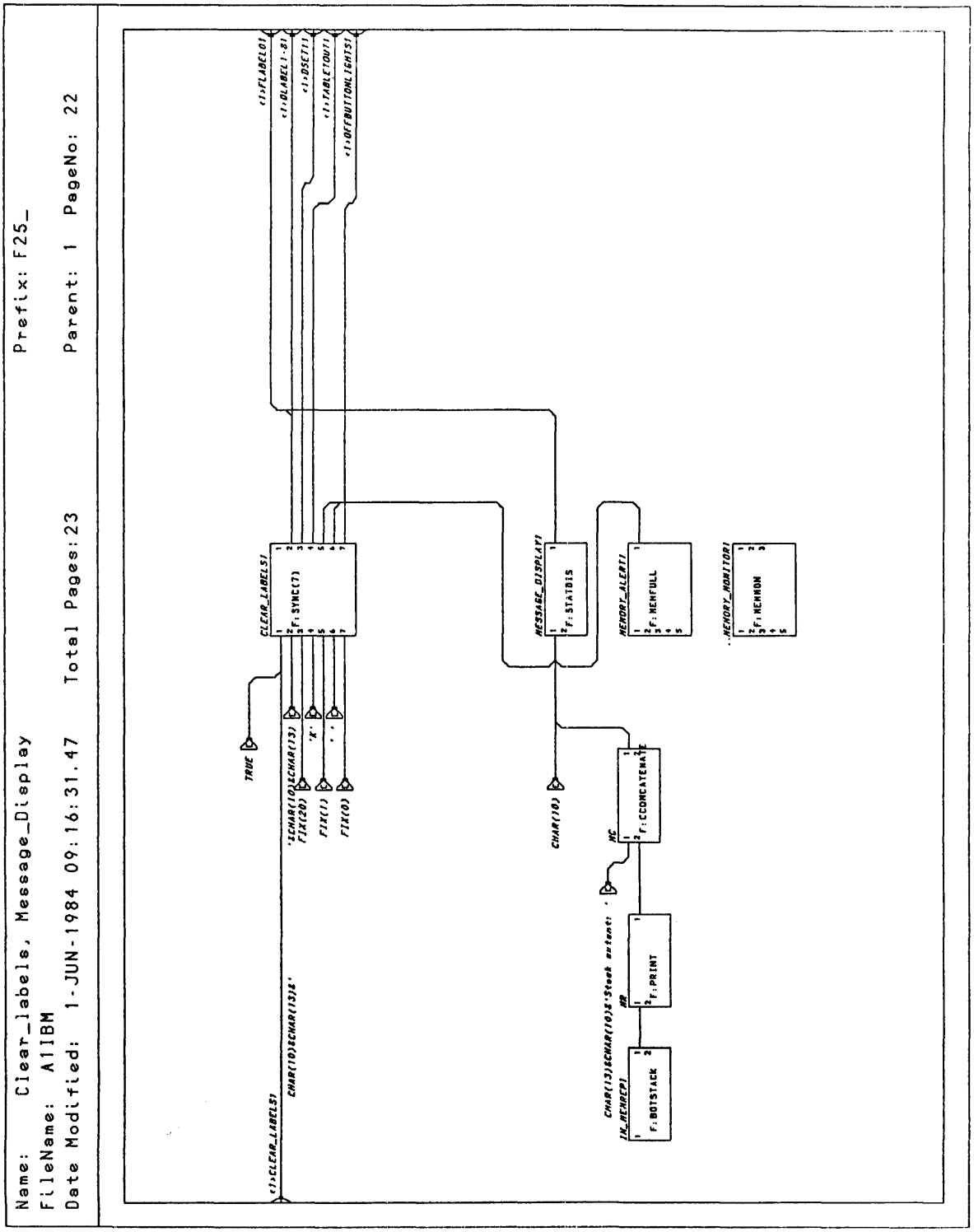


Figure 8-48. PS 390 Host Input Data Flow (IBM)

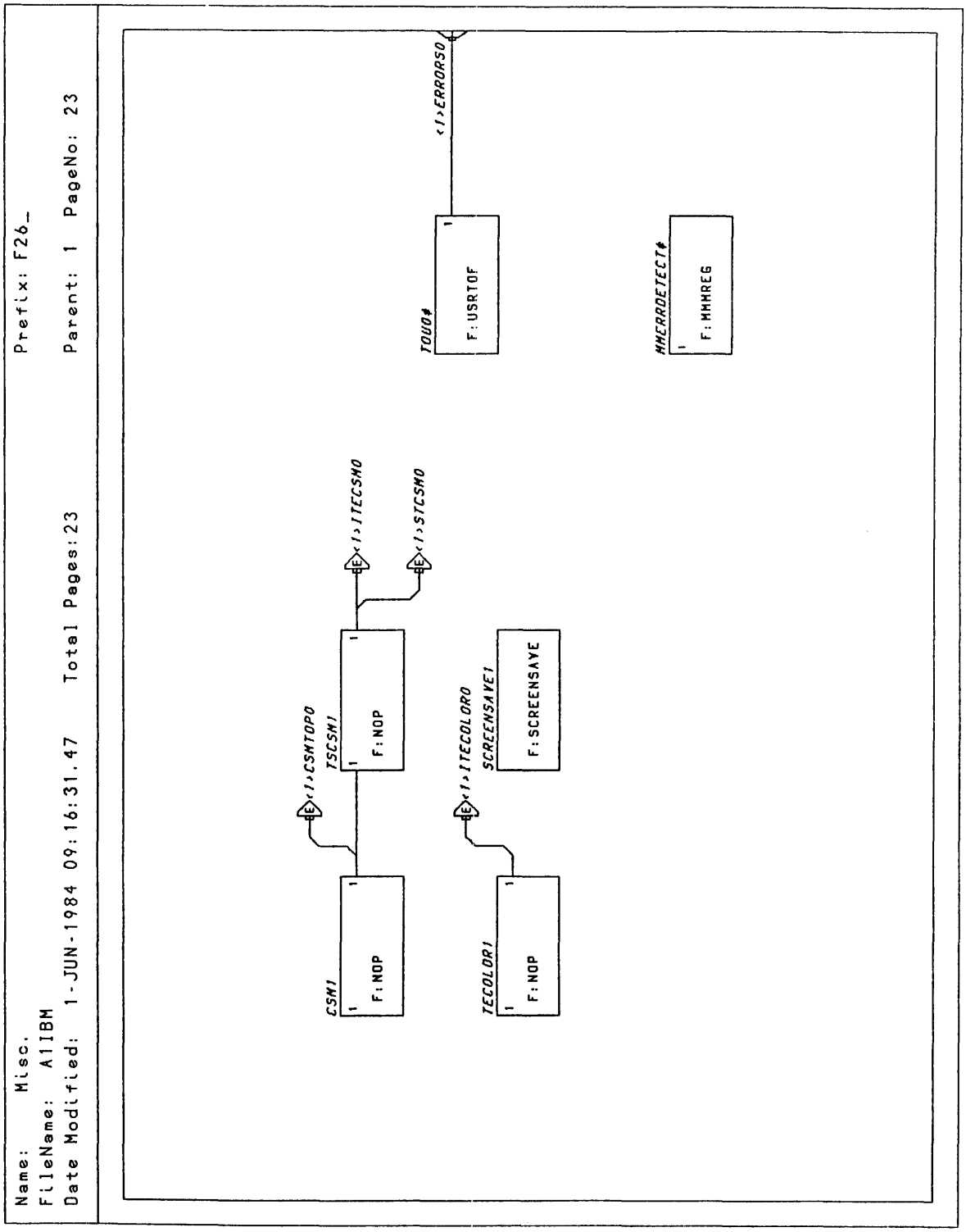


Figure 8-49. PS 390 Host Input Data Flow (IBM)

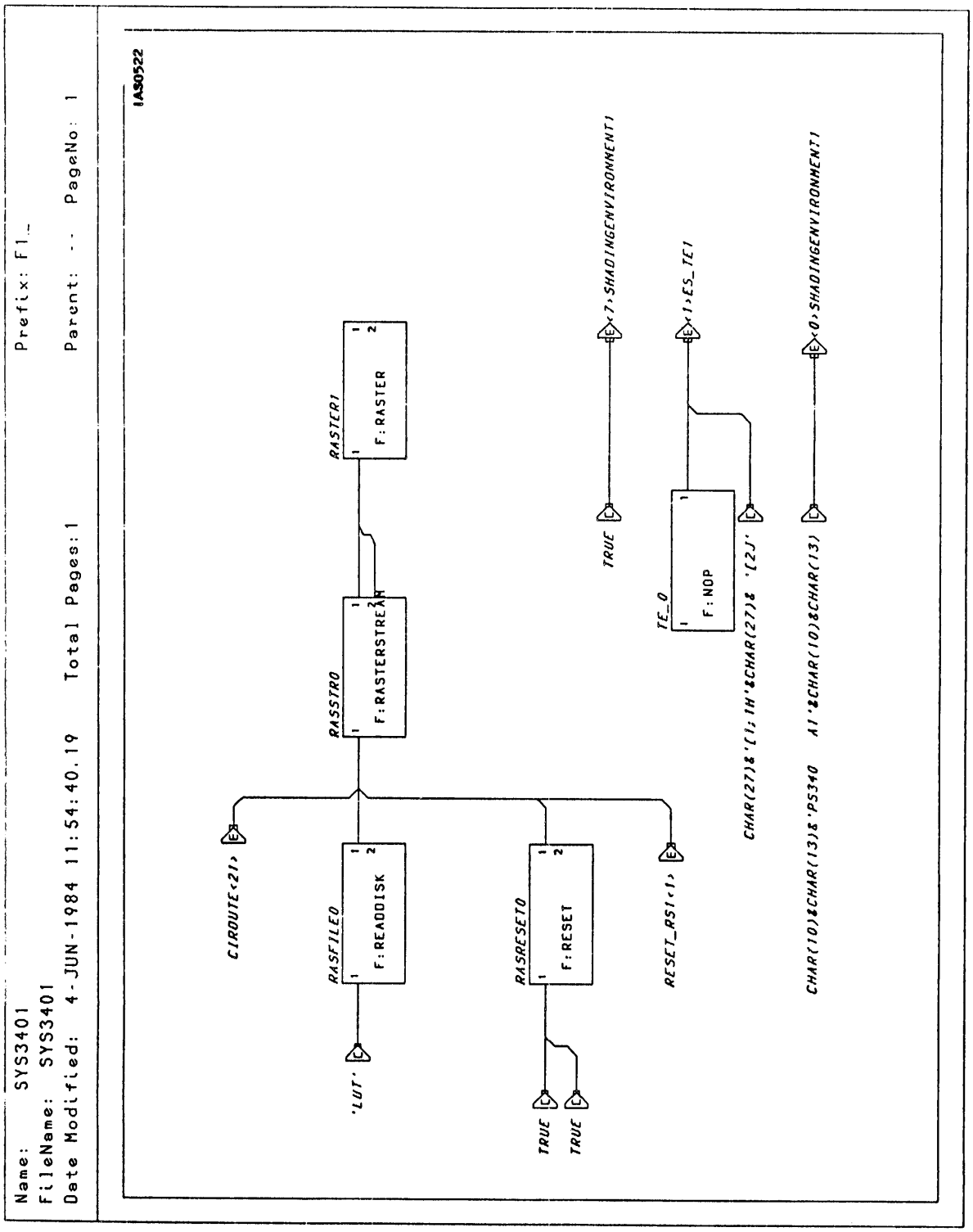


Figure 8-50. Raster System Host Input Data Flow (DEC)

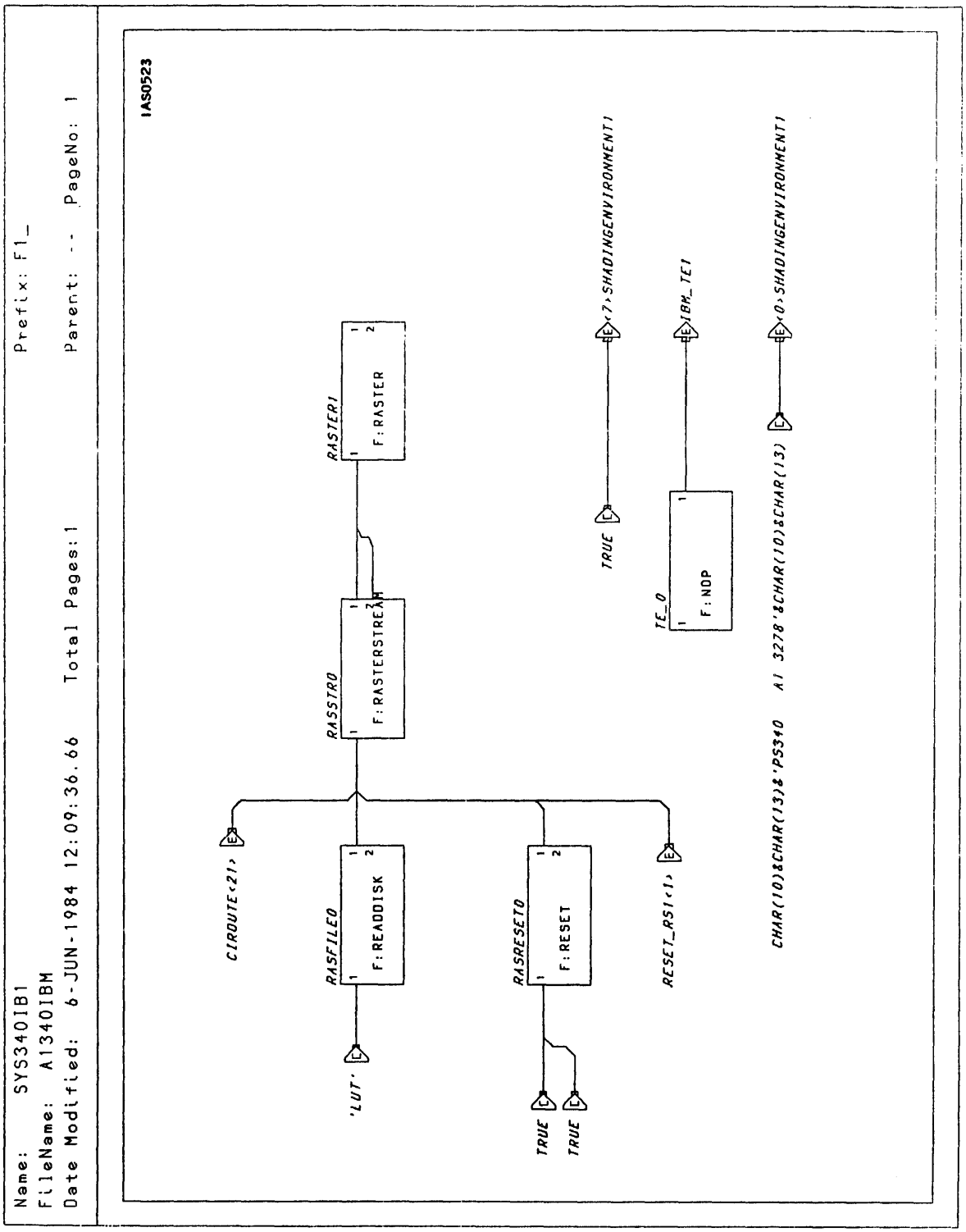


Figure 8-51. Raster System Host Input Data Flow (IBM)

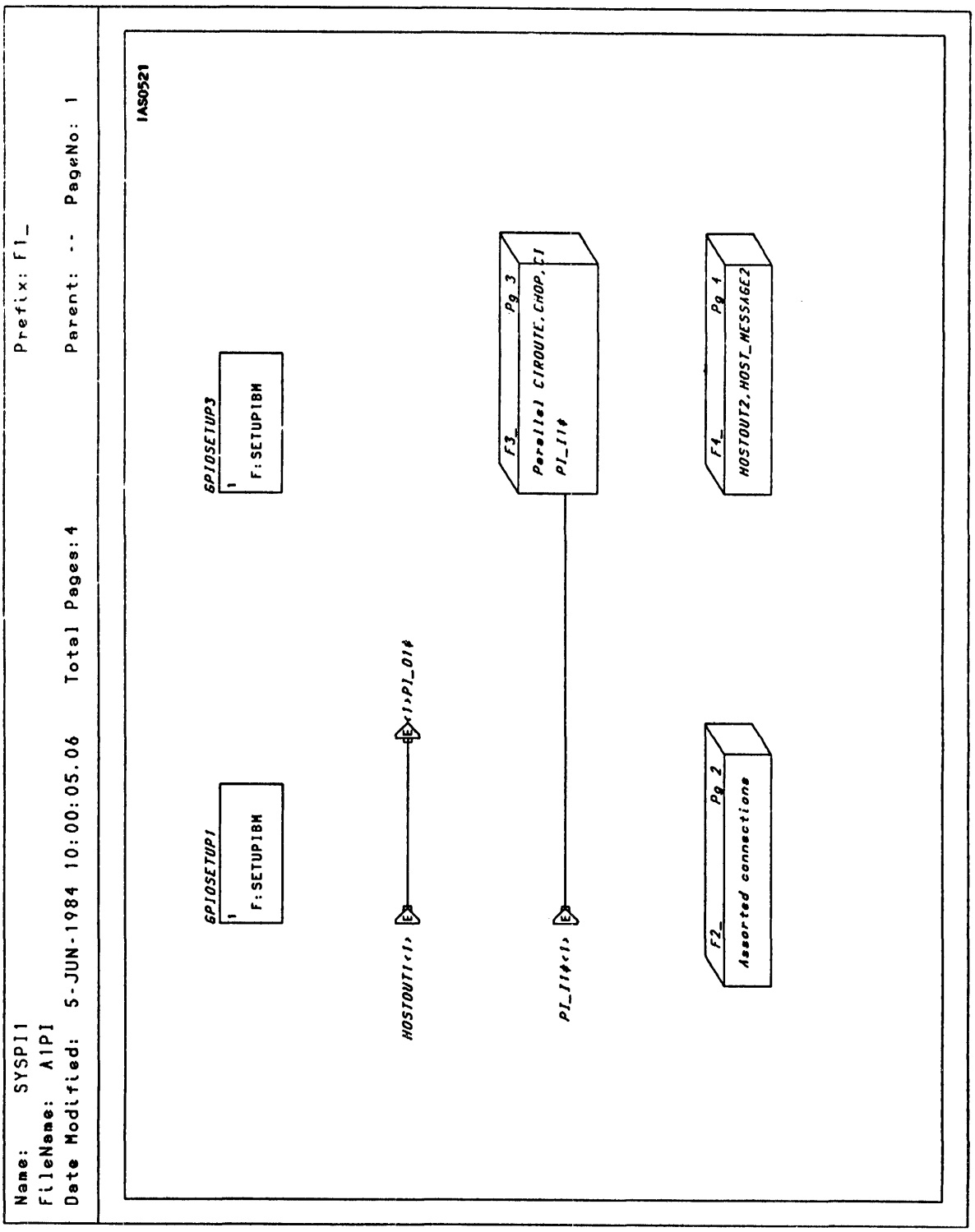


Figure 8-52. DEC Parallel Interface Host Input Data Flow

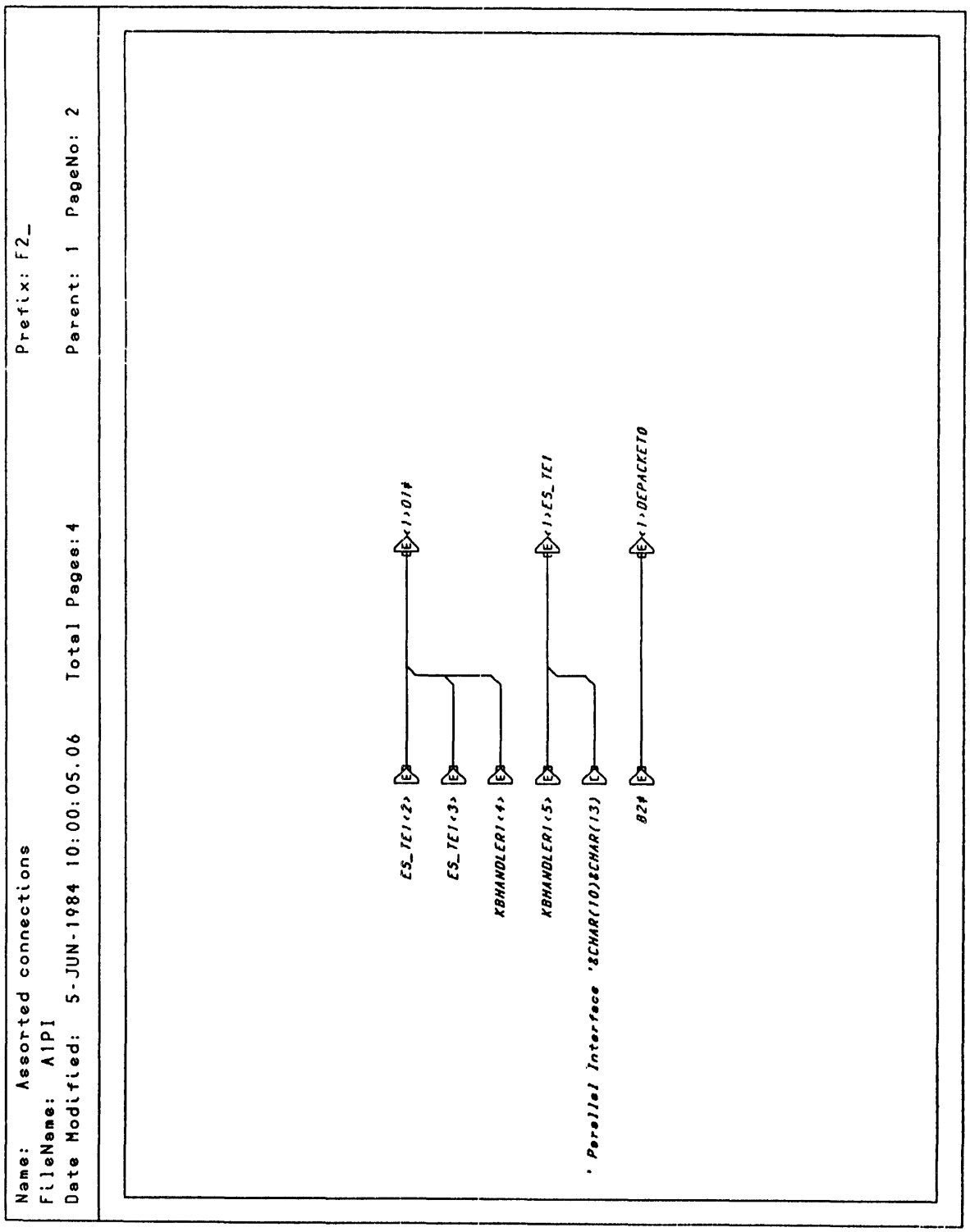


Figure 8-53. DEC Parallel Interface Host Input Data Flow

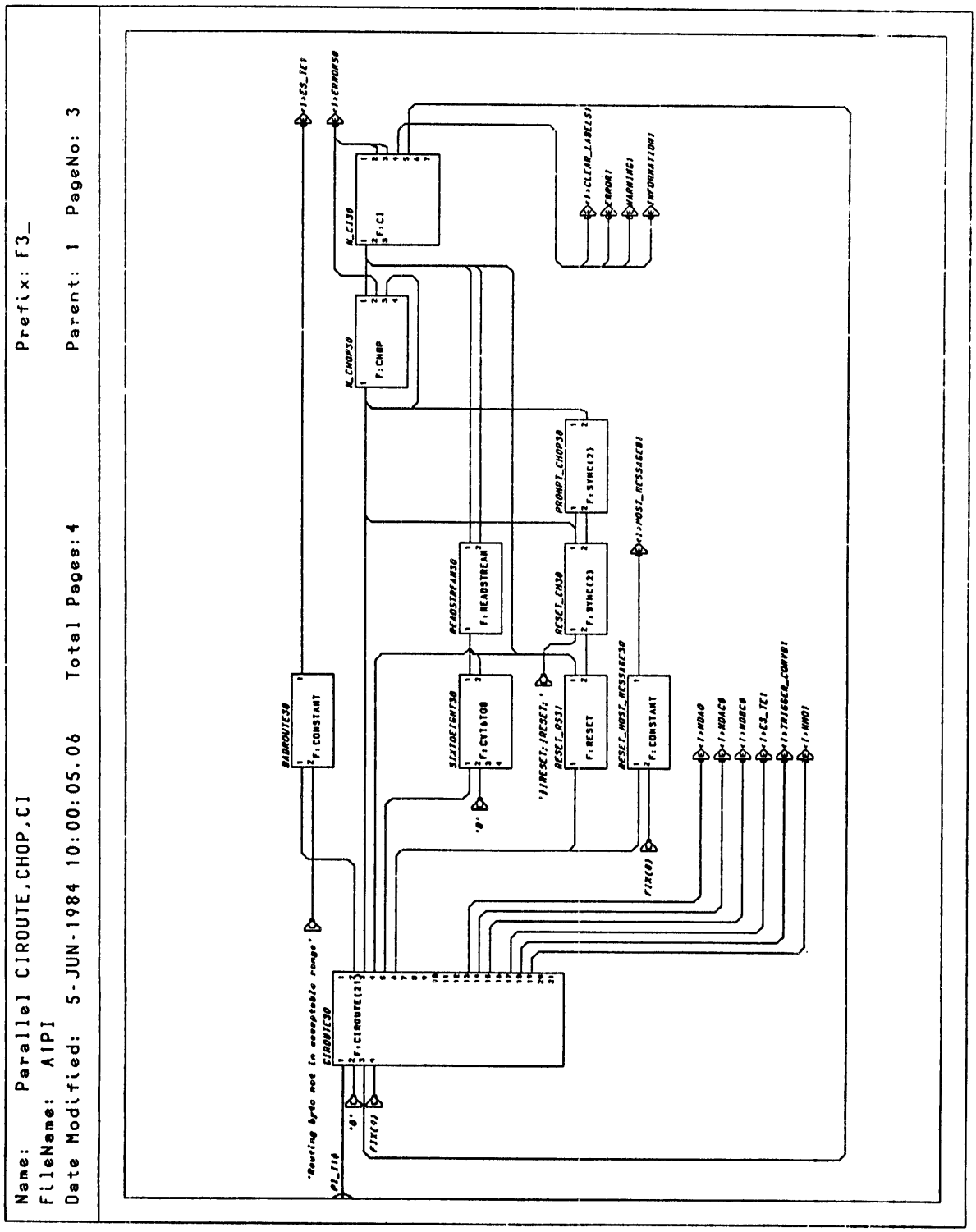


Figure 8-54. DEC Parallel Interface Host Input Data Flow

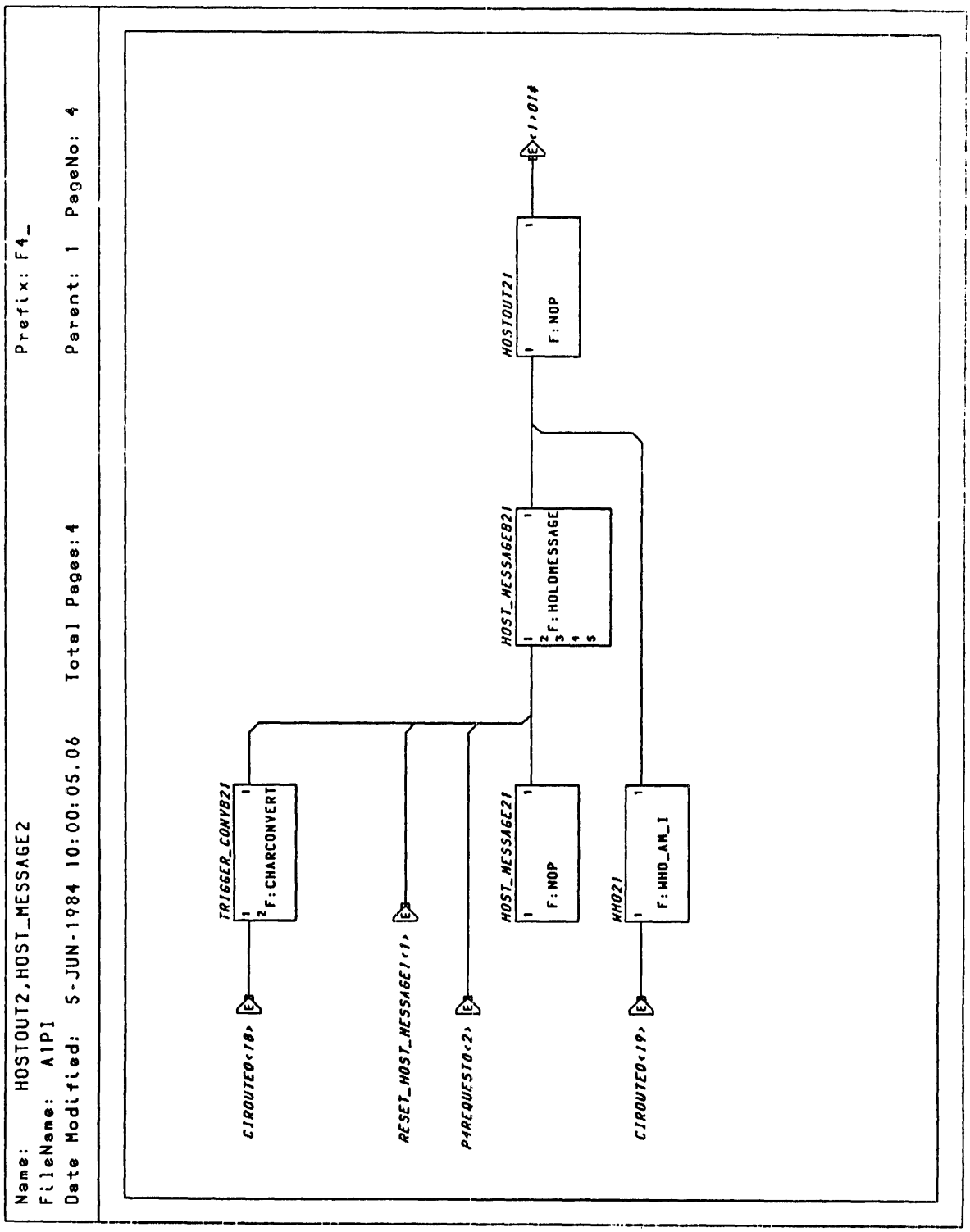


Figure 8-55. DEC Parallel Interface Host Input Data Flow

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 Instances*.

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:

```
SC0$ := 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 CSMTOP0
CSMTOP0 := SET CSM OFF
   THEN GT0$
 SEND TRUE TO <-1>HVP1$
GT0$ := INSTANCE OF GVP0$, TVP0$, MD0$
```

Graphics Display Structure:

```
GVP0$ := VIEW HORIZONTAL = -1:1 VERTICAL = -1:1 INTENSITY = 0:1
   THEN PICK_LOCATION1;
PICK_LOCATION1: := SET PICK LOCATION = 0,0 .01, .01
   THEN GCUR0$;
GCUR0$ := INSTANCE WB$1, CT1$,
WB$! := WRITEBACK
   THEN GD0$;        {All Display Commands append to GD0$}
CT1$ := 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:

```
TVP0$ := VIEW HORIZONTAL = -1:1 VERTICAL = -1:0
   THEN TENOSLAVE0$;
TENOSLAVE0$ := SET DISP ALL ON
   THEN TECSM0;
TECSM0 := SET CSM OFF
   THEN TECOLOR0;
TECOLOR0 := SET COLOR 240.0, 1.0
   THEN TD0$; {The Terminal Screen is appended to TD0$}
```

Terminal Emulator Display Structure for IBM 3278:

```
IVPO$ := CHAR FONT IMBFONT$
   THEN ITENOSLAVE0$;
ITENOSLAVE0$ := SET DISP ALL ON
   THEN ITECSM0;
 ITECSM0 := SET CSM OFF
   THEN ITECOLOR0;
ITECOLOR0 := SET COLOR 240, 1
   THEN IBMSCR0$;
IBMSCR0$ := INSTANCE IBMSC0$, IBMLINE$;
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 C0$;
    IF LEVEL =  1 THEN C1$;
    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$:=CHAR 'Unknown crash';
C0$:=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:

```
SVP0$ := VIEW HORIZONTAL= -1:1 VERTICAL = -1:1
    THEN SZ0$;
STCSM0 := SET CSM OFF
    THEN SS0$;
SS0$ := CHAR SCALE 0.03
    THEN SS0$;
SS0$ := 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   ';
SA0$ := CHAR -1,.0 'Press special key to be breakkey, F1 to exit. ';
SB0$ := 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

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>[Pn*l*

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.

ANSI SEQUENCES	DESCRIPTION
<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.

ANSI SEQUENCES	DESCRIPTION
<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>Ol
	.		<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>[0J 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>[0K 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.

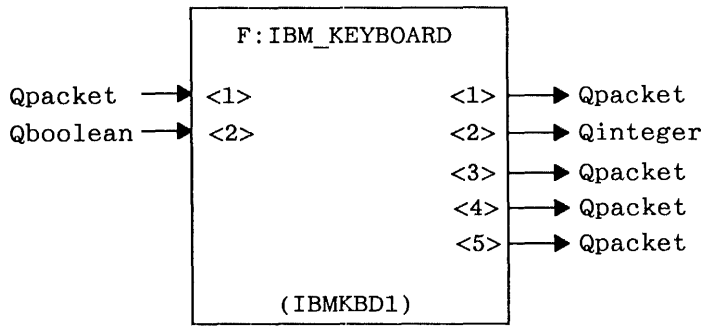


Figure 10-1. F:IBM_KEYBOARD

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:

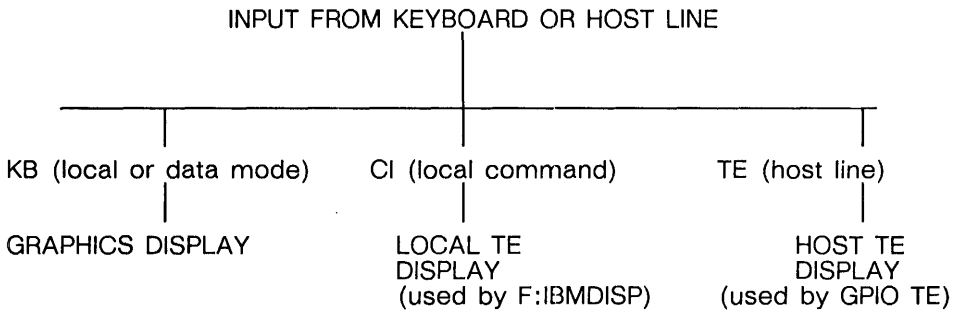
```
SEND TRUE TO <2>IBMKBD1;
```

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:



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.

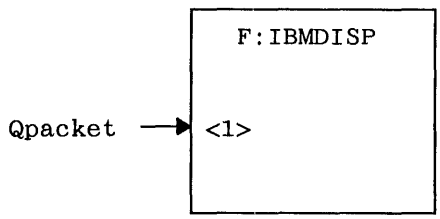


Figure 10-2. F:IBMDISP

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.

ɢ 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

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.*

RM13. INTERACTIVE DEVICES

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.

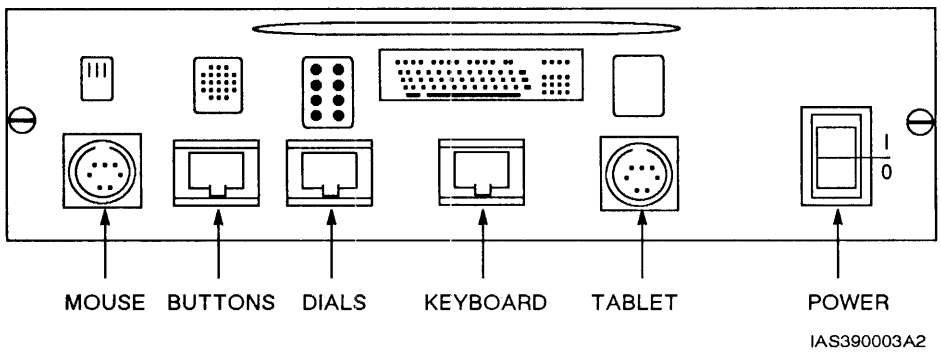


Figure 13A-1. Front Panel of Peripheral Multiplexer

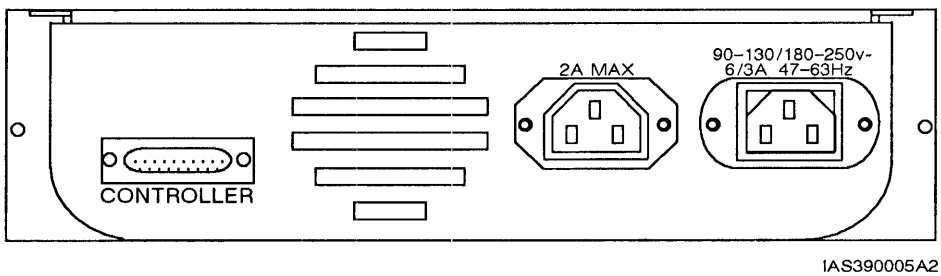


Figure 13A-2. Back Panel of Peripheral Multiplexer

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 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'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		SHIFT+KEY		CTRL+KEY	
	CODE	CHARACTER	CODE	CHARACTER	CODE	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'0D'	CR	X'0D'	CR	X'0D'	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.

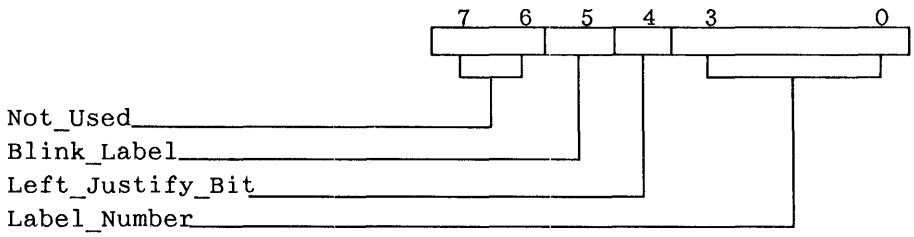
All data transmitted to the LEDs for display in line mode must be terminated with a RETURN character. After a RETURN character is entered, the display is cleared when the next valid character is received. The received character is output to the leftmost LED character, and the LEDs are filled left to 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'0D'	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'0D'	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'0D'	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

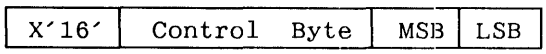
The control dials unit outputs pulses when any of the dials are turned. From the pulses, the control dials interface determines:

- 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:



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

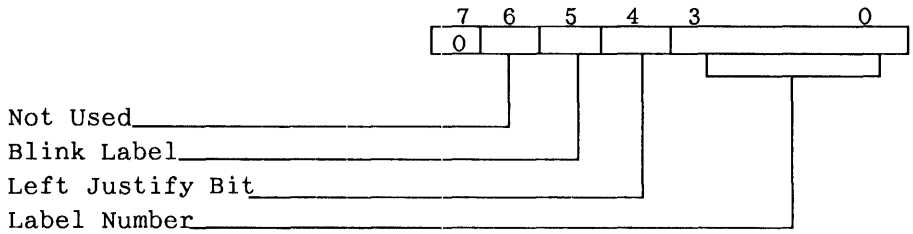
3.3 LED Display Operation

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.

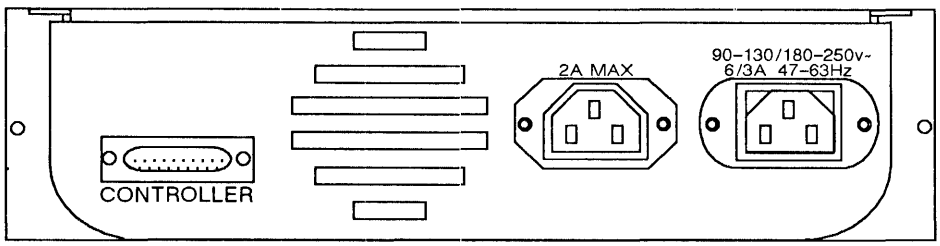


Figure 13B-1. Backside Connectors for the Peripheral Multiplexer

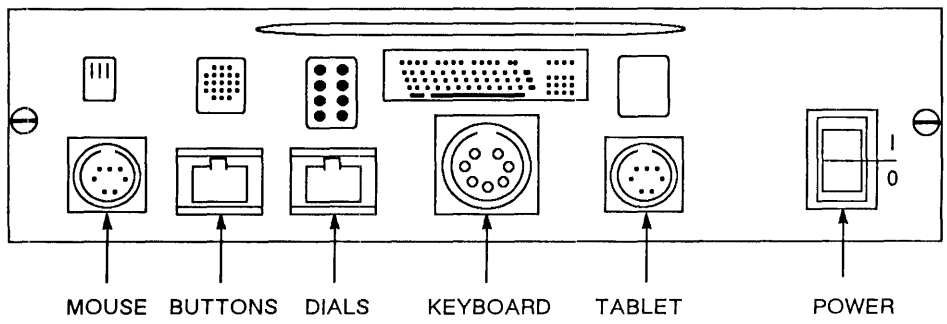


Figure 13B-2. Connectors for the PS 390 Style Interactive Devices

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.

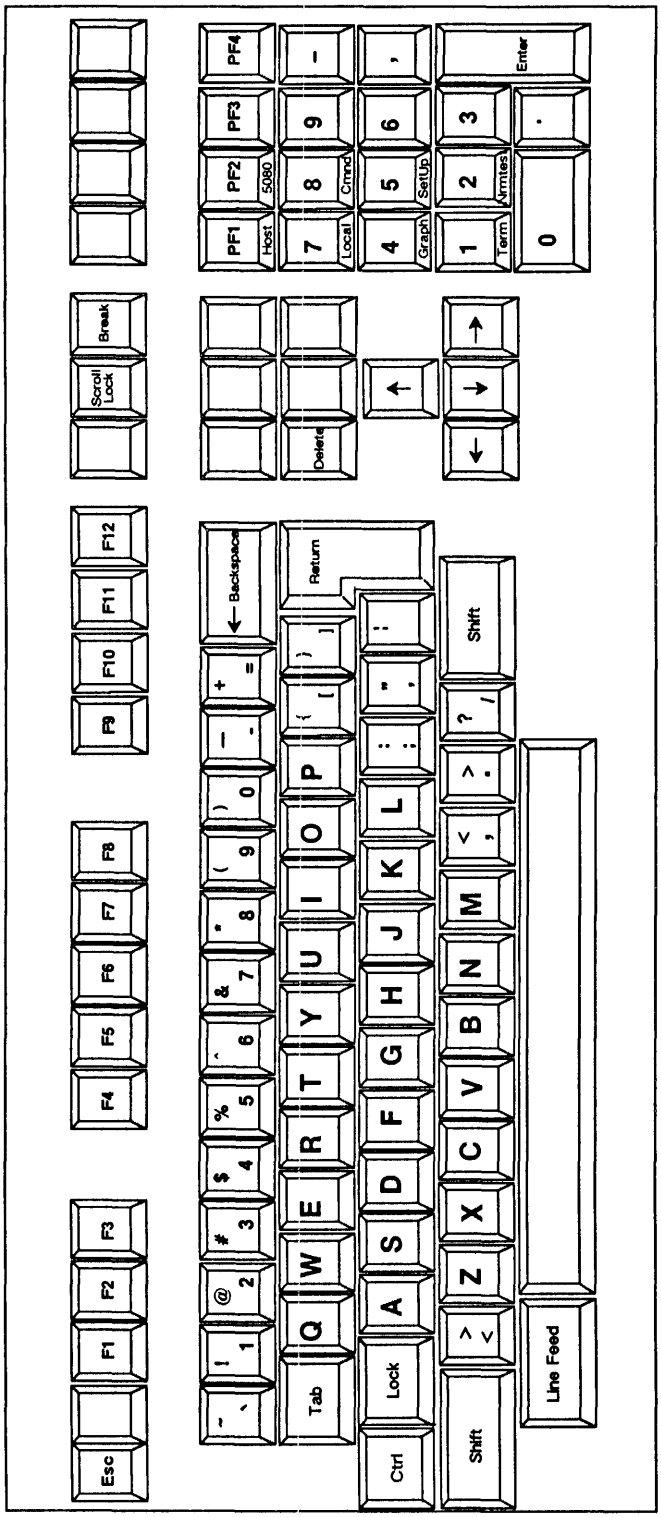


Figure 13B-3. The PS 390 Style Keyboard

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 Label	*Key Alone*		*SHIFT+Key*		*CTRL+Key*	
	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 Label	*Key Alone*		*SHIFT+Key*		*CTRL+Key*	
	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	*Key Alone*		*SHIFT+Key*		*CTRL+Key*	
Label	*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 Label	*Key Alone Code*	*SHIFT+Key Code*	*CTRL+Key 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.

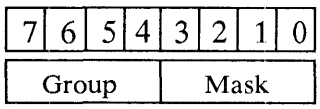


Table 13B-11. Function Button Light Control Message Byte

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.

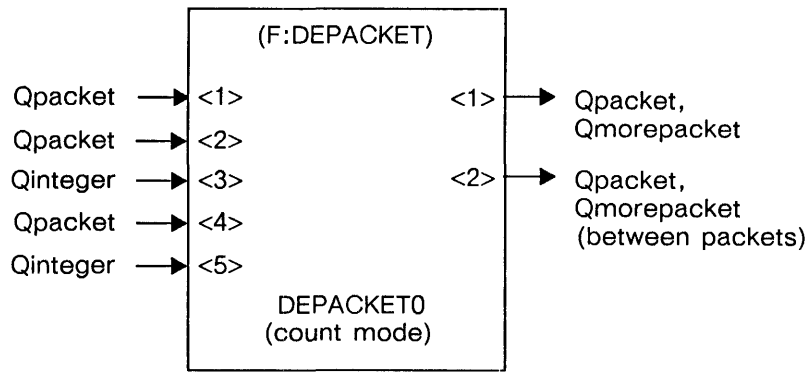
The CI expects tokens that consist of a size, a data type, and a value. Once given, the type of command is implicit in the type of the token, such as "Qsetcontrast" for "Set Contrast." The CI accepts tokens until it has enough to carry out a command.

1.1. Routing Functions

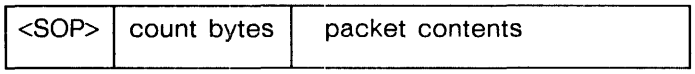
Data is sent from the host to the PS 390 as a stream of bytes. The bytes contain information that tells the PS 390 intrinsic functions the nature of the message and where it is to be sent internally. The following is a list of the data transfer modes used in host/PS 390 communication and a brief description 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:



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.

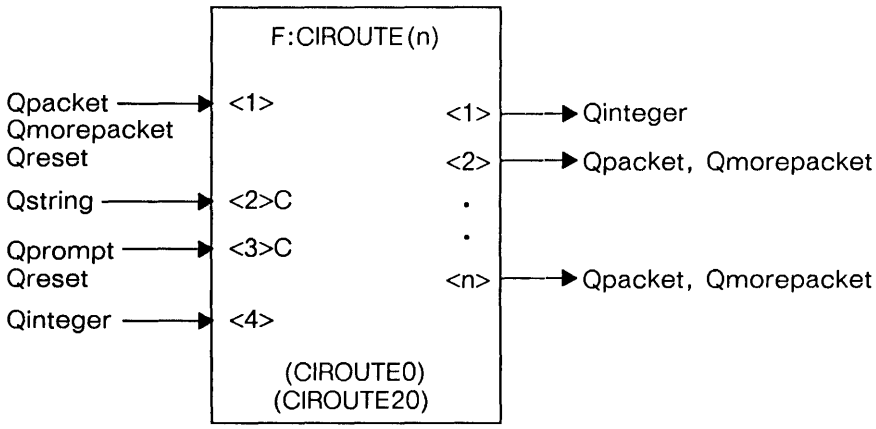
F:CIROUTE(n)

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.

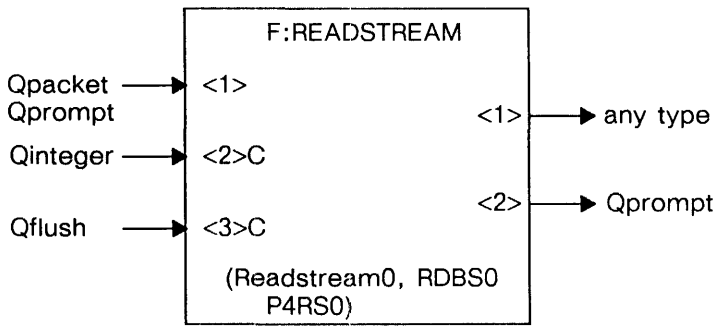
When instancing this function, a parameter is required to specify the number of outputs.

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.

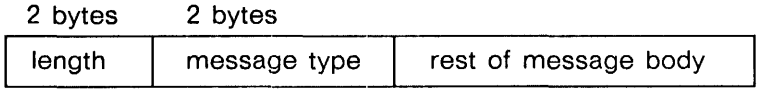
The definitions for the inputs and outputs for F:CIROUTE(n), and routing bytes used by F:CIROUTE(n) are described in Section *RM2, Intrinsic Functions*.

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.

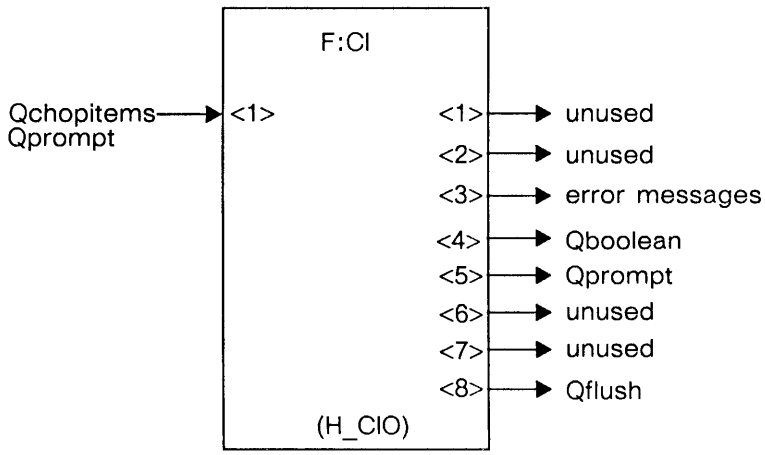


This function converts an 8-bit stream into arbitrary messages. It takes two bytes as the count of information (including message type) and creates a message of that size with the bytes of information that follow it. The message format on input <1> is:



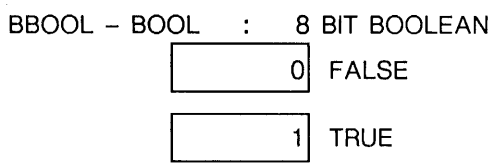
F:CI

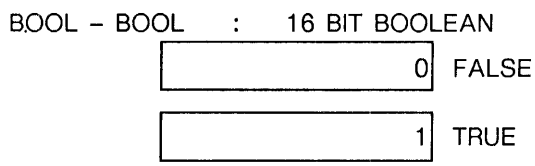
The CI accepts messages from the GSRs through an instance of F:READSTREAM.



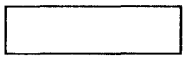
This function interprets commands, creating display structures and function networks. It receives input either from a chop/parse function or a READSTREAM function (if using the GSRs).

1.2. Data Formats for Data Types

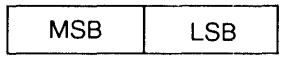




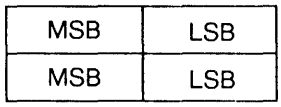
INT8 - BYTE : 8 BIT INTEGER



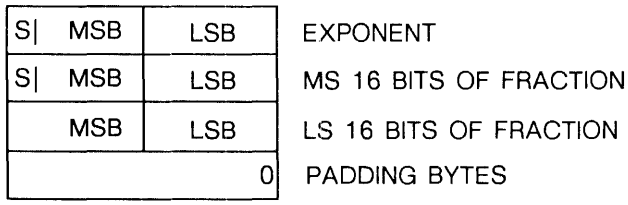
INT16 - WORD : 16 BIT INTEGER



INT32 - LWORD : 32 BIT INTEGER



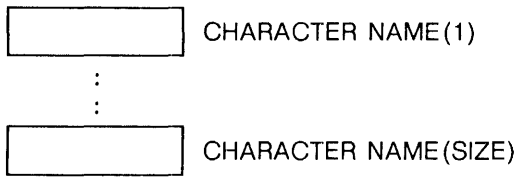
PSREAL - REAL32 : 64 BIT REAL



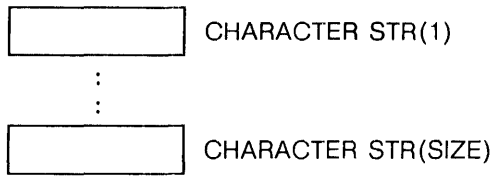
NOTE

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



STRING - NAME, 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
```

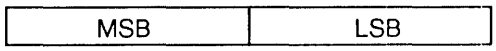
```
HANDLE := XFORM MATRIX (APPLIED TO APPLY);

       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:



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.

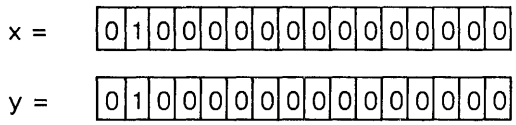
CAUTION

The carriage control characters must be suppressed when transmitting binary data, or the carriage control characters will be interpreted as binary data.

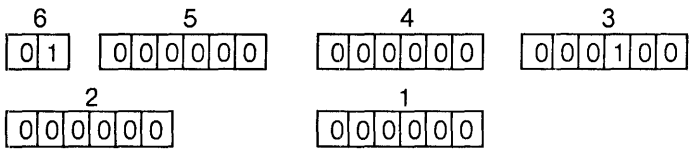
2. A two-word set is broken up into five bytes with six significant bits, and one byte with two significant bits. These bits are extracted from the least significant end to the most significant end of the two-word set.

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.

	Two significant bits
byte6	0 0 1 1 0 0 0 1
	Six significant bits
byte5	0 0 1 1 0 0 0 0
	Six significant bits
byte4	0 0 1 1 0 0 0 0
	Six significant bits
byte3	0 0 1 1 0 1 0 0
	Six significant bits
byte2	0 0 1 1 0 0 0 0
	Six significant bits
byte1	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;@2D0300000000000000000000000000000000000000000000000000
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.

RM15. RELEASE NOTES

RM16. INDEX

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."

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z