TAM-1001-007

DMS-100 Family

PMIST

Technical Assistance Manual

BCS36 and up Standard 08.02 December 1993

northern telecom

DMS-100 Family

PMIST

Technical Assistance Manual

Publication number: TAM-1001-007
Product release: BCS36 and up
Document release: Standard 08.02
Date: December 1993

Publication history

December 1993

BCS36 Standard 08.02 release of this document changed various commands, parameters, and variables

October 1993

BCS36 Preliminary 08.01 release of this document to verification office

March 1993

BCS35 Standard 07.01 changed command format and made changes to various commands, parameters, and variables

September 1990

BCS31 Standard 06.02 restructured document

Contents

List of figures

List of tables

About this document

This Technical Assistance Manual (TAM) provides an introduction to the peripheral module intercept system test (PMIST). It includes instructions for using PMIST and an introduction to interpreting PMIST output. It is intended for use by skilled maintenance personnel that have a thorough knowledge of the DMS-100 software and experience in maintaining the DMS-100 switch.

When to use this document

Northern Telecom (NT) software releases are referred to as batch change supplements (BCS) and are identified by a number, for example, BCS29. This document is written for DMS-100 Family offices that have BCS36 and up.

More than one version of this document may exist. The version and issue are indicated throughout the document, for example, 01.01. The first two digits increase by one each time the document content is changed to support new BCS-related developments. For example, the first release of a document is 01.01, and the next release of the document in a subsequent BCS is 02.01. The second two digits increase by one each time a document is revised and rereleased for the same BCS.

To determine which version of this document applies to the BCS in your office, check the release information in *DMS-100 Family Guide to Northern Telecom Publications*, 297-1001-001.

How to identify the software in your office

The *Office Feature Record (D190)* identifies the current BCS level and the NT feature packages in your switch. You can list a specific feature package or patch on the MAP (maintenance and administration position) terminal by typing

>PATCHER;INFORM LIST identifier

and pressing the Enter key.

where

identifier is the number of the feature package or patch ID

You can identify your current BCS level and print a list of all the feature packages and patches in your switch by performing the following steps. First, direct the terminal response to the desired printer by typing

>SEND printer_id

and pressing the Enter key.

where

printer_id is the number of the printer where you want to print the data

Then, print the desired information by typing

>PATCHER;INFORM LIST;LEAVE

and pressing the Enter key.

Finally, redirect the display back to the terminal by typing

>SEND PREVIOUS

and pressing the Enter key.

Where to find information

The chart below lists the documents that you require to understand the content of this document, or to perform the tasks it describes. These documents are also referred to in the appropriate places in the text.

More than one version of these documents may exist. To determine which version of a document applies to the BCS in your office, check the release information in *DMS-100 Family Guide to Northern Telecom Publications*, 297-1001-001.

Number	Title
297-1001-100	*System Description*
297-1001-103	*Peripheral Modules*
TAM-1001-000	*Index of Technical Assistance Manuals*
297-1001-001	*Master Index of Practices*
297-1001-006	*Maintenance System Description*
297-1001-007	*Maintenance and Administration Tools*
297-1001-010	*Maintenance and Administration Position*
297-1001-129	*Input/Output System Reference Manual*
297-1001-510	*Log Report Manual*
297-1001-515	*Peripheral Modules Maintenance Reference Manual*

What precautionary messages mean

Danger, warning, and caution messages in this document indicate potential risks. These messages and their meanings are listed in the following chart.

Message	Significance
DANGER	Possibility of personal injury
WARNING	Possibility of equipment damage
CAUTION	Possibility of service interruption or degradation

Examples of the precautionary messages follow.

DANGER
Risk of electrocution
The inverter contains high voltage lines. Do not open the front panel of the inverter unless fuses F1, F2, and F3 have been removed first. Until these fuses are removed, the high voltage lines inside the inverter are active, and you risk being electrocuted.

WARNING
Damage to backplane connector pins
Use light thumb pressure to align the card with the connectors. Next, use the levers to seat the card into the connectors. Failure to align the card first may result in bending of the backplane connector pins.

CAUTION
Loss of service
Subscriber service will be lost if you accidentally remove a card from the active unit of the peripheral module (PM). Before continuing, confirm that you are removing the card from the inactive unit of the PM.

How commands, parameters, and responses are represented in command descriptions

Two command conventions exist. They include the following.

- command expansion - representations of commands including all parameters, variables and syntactic characteristics
- command example - representations of commands as they are entered

Command expansion conventions

A command table is used for a command expansion. This table consists of the following sections:

- the command expansion, which contains
  - all parameters
  - all variables
  - hierarchy (the order in which elements must be entered)
  - syntax
  - truncated and abbreviated forms when allowed
  - defaults
- the parameter and variable descriptions. This section follows the command expansion and contains an alphabetical listing of all parameters and variables with a description of each.

Command elements are represented exactly as they are entered, except when Italic font is used to indicate that an element is a variable name or a certain default.

Commands

The command is represented in bold type. When commands are not case-sensitive, they are in lowercase.

The command appears to the left of all other elements (parameters and variables).

When truncated or abbreviated forms of a command are allowed, they appear directly beneath the long form of the command.

Parameters

Parameters are represented in unbolded type. When parameters are not case-sensitive, they are in lowercase.

Variables

Variables are represented in italics. Italics indicates that the variable, as represented, is not entered, but replaced with an element, a value, range, number, or item from a list.

The numbers, values, ranges, and lists are described in detail for each variable in the parameters and variables description section below the expansion.

Hierarchy

The order in which command elements are entered is represented by their order of appearance, from left to right. When several elements appear in a vertical list, only one of them may be selected for that position.

Defaults

A default parameter is underlined.

The system takes a default action when an element in a vertical list is not required and it is usually an action indicated by one of the elements that can be selected. Occasionally, the default action is something different than one indicated. These nonselectable defaults are represented by the word, *default*, in italics, to indicate that it is never entered. The default is then described in the parameters and variables section.

Related groups of elements

When an element is directly followed by another element, the second element is required when the first element is selected.

To distinguish which elements relate to which, brackets surround those elements that, as a group, pertain to other elements. Only those elements that horizontally directly precede or follow the brackets are related to the elements within the brackets. When elements are not in brackets, only those elements that directly precede or follow them are related.

The following is an example of a command expansion.

bsy command parameters and variables

Command	**Parameters and variables**			
bsy	[link	*ps_link*]	*noforce*	*wait*
	pm		force	nowait
	unit	*unit_no*		

Parameters and variables	Description
force	overrides all other commands and states in effect on the specified units. If the whole PM is to be taken out of service, confirmation, yes or no, is required.
link	busies one of the P-side links specified by *ps_link.*
noforce	indicates default condition when *force* is not entered.
nowait	allows the MAP terminal to be used for other command entries before bsy force is confirmed. Nowait is used only with force.

-continued-

bsy command parameters and variables (continued)

Parameters and variables	Description
pm	busies both units of the pm.
ps_link	specifies which of the P-side links is to be busied. Range is 0 through 3.
unit	busies one unit of the pm specified by *unit_no.*
unit_no	specifies which unit of the pm is to be busied. Range is 0 through 1.
wait	indicates default condition when *nowait* is not entered.

End

Command examples

Command examples use the same conventions as a command expansion, except that all command elements are bold and are entered just as represented. If the variable is shown with a value, it is entered exactly like a command or parameter. If the variable name is used, it is in bold italics to indicate that it is not entered as represented. The following two examples illustrate this difference.

- This is a command example containing a variable name.

  bsy link *ps_link*
  and pressing the Enter key.

- This is a command example containing a variable value.

  bsy link 2
  and pressing the Enter key.

The PMIST utility

This section describes the peripheral module intercept system test (PMIST) subsystem. The description includes PMIST uses, restrictions, and limitations.

Purpose of PMIST

The PMIST is a low-level internal diagnostic tool that records messages sent between the central control (CC) and the peripheral modules (PM) in the DMS-100 and DMS-200 systems. PMIST monitors and records incoming and outgoing messages to help determine whether the CC is responding properly to PM signals and whether a PM is processing CC instructions correctly.

Messages sent from a PM to the CC are incoming messages, or reports. Instructions sent from the CC to a PM are outgoing messages.

I/O messages are composed of a message type and a message body. The CC uses the message type to determine what has occurred, such as an off-hook condition or digit reporting. The message body contains information about the occurrence; for example, digit reporting must contain digit information.

PMIST is capable of performing the following:

- recording what I/O messages are sent to and from the CC
- inserting user-specified I/O messages
- converting node to name (for example, node number 16 = TM8 1) for any node
- storing recorded messages in a file

PMIST subsystem

PMIST is a command interpreter (CI) command that can be entered from any level of the maintenance and administration position (MAP). The PMIST command accesses the PMIST subsystem which responds by displaying the PMIST MULTI USER prompt. The subsystem thus entered allows you to use any of the PMIST commands. PMIST can be accessed with a password.

The PMIST subsystem consists of the following five modules:

- PMIST
  implements the PMIST subcommands and establishes the user interface for PMIST. The Command Interpreter (CI) access level is extended to include this program increment.
- IMD
  implements the Intercepted Message Dispatcher (IMD) process that receives, logs, records, and dispatches intercepted I/O messages.
- MIDC
  initializes the PMIST subsystem and implements the Message Intercept and Dispatch Control (MIDC) utility. MIDC coordinates activities between the user and the IMD process. MIDC makes use of the I/O message insertion and intercept "hooks" in the I/O system.
- DIOMSG
  implements a procedure to format and display an I/O message. The I/O messages are mostly symbolic.
- PMISTUTL
  implements the CONVERT facility.

When PMIST is used

PMIST can be used to validate call processing and other message-driven software and to verify PM operation.

Conditions that may require the use of PMIST include:

- call failure (other than a trap)
- peripheral maintenance problems
- PM, Network, Input/Output Controller (IOC) maintenance problems
- improper messages sent to or from the CC
- hardware problems within peripherals
- PM not interpreting execs and primitives correctly.

Restrictions and limitations

The PMIST debugging tool can cause a switch to fail if used incorrectly. PMIST is not recommended for use in live DMS-100/200 offices. If PMIST must be used in a working office, operating company personnel should be aware that a restart may be required if problems occur.

The SELECT subcommand should always be set to ON.

PMIST has the potential to monitor 40 nodes at one time in a lab. However, no more than three nodes should be used in a working office due to switch degradation.

Real time effect

The real time estimates provided in this section represent the sum of the following factors:

- the percentage of real time used by Call Processing Occupancy (CPOCC) as overhead

  Note: Call Processing Occupancy is the proportion of CPU time used for call processing.

- the percentage of real time used by CPOCC per call traced
- the percentage of real time used by priority 6 (foreground) processes.

PMIST Real time Impact

The PMIST utility causes 2.4% degradation of the total CPU time. This total is based on 1200 Calls Per Hour (CPH) being traced with RECORD set to OFF and the switch processing 180,000 CPH.

XPMIST Real time Impact

The XPMIST utility has the following impact on switch performance:

RECORD ON, tracing 240 CPH	4.9%
RECORD ON, INCLUDENODE, tracing 4000 CPH	9.7%
RECORD ON, ASSOCIATE ON, tracing 1200 CPH	13.0%
All commands on, tracing 4000 CPH	20.6%

PMIST user classes

Three different classes of PMIST users are defined:

- single user
- multi-user
- observer

Single user class

You are classed as a single user if you are the only person using PMIST and you enter either `INTERCEPT ON`°°, `SELECT OFF`°°, `IMMUNE`, `VERIFY`, or `RESTART`. The PMIST prompt will reflect the change of user status by displaying PMIST SINGLE USER. Users who enter PMIST once a single user is operating PMIST will be treated as observers.

The single user can use all PMIST subcommands and operations.

Multi-user class

When you enter the PMIST subsystem, you are classified as a multi-user. The PMIST prompt will reflect the user status by displaying PMIST MULTI USER.

Up to five multi-users can monitor and record PMIST messages at one time. If a sixth user tries to enter PMIST, access is denied and the message ERROR--MAXIMUM NUMBER OF USERS IN PMIST appears.

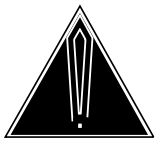
Multi-users can use all the PMIST subcommands except:

- INTERCEPT ON
- SELECT OFF
- IMMUNE
- VERIFY
- RESTART.

Usage notes

1 No more than three nodes should be monitored by all users at one time on a live switch because of real time constraints.

2 Injecting user-defined messages into the I/O system affects other users in the multi-user environment.



CAUTION
Commands intercept on and select off can cause the switch to fail.
Refer to page NO TAG for more information on the intercept command and to page NO TAG for the select command.

Observer class

Users that enter the PMIST subsystem after a single user has been classified are treated as observers. Observer status is reflected by the PMIST OBSERVER prompt.

Observers can process only the following subcommands:

- CONVERT
- PMIST
- INSERT subcommands
- DISPLAY
- EXTRACT
- NODENO
- DEVICENAME

- QUIT.

Observers cannot monitor or record messages.

Informative references

The listed publications provide a foundation for understanding a broad scope of information surrounding the PMIST utility. These publications are not referenced within the text of this document.

> ***Note:*** The documents listed may exist in more than one version. See 297-1001-001 to determine the release code of the version compatible with a specific release of software.

Number	Title
297-1001-100	*System Description*
297-1001-103	*Peripheral Modules*
TAM-1001-000	*Index of Technical Assistance Manuals*
297-1001-001	*Master Index of Practices*
297-1001-006	*Maintenance System Description*
297-1001-007	*Maintenance and Administration Tools*
297-1001-010	*Maintenance and Administration Position*
297-1001-129	*Input/Output System Reference Manual*
297-1001-510	*Log Report Manual*
297-1001-515	*Peripheral Modules Maintenance Reference Manual*

Determining terminal and node numbers

This part describes what node and terminal numbers are, what terminal 0 does, and how to calculate node and terminal numbers.

Node and terminal numbers

Message tracing requires identifying the terminals that need to be monitored. A terminal is an external connection to the DMS-100, such as a line, a trunk, or a data link. PMIST recognizes which terminal to monitor by the terminal identifier entered by the PMIST user.

A terminal identifier is composed of a node number and a terminal number.

A node is any unit that can accept or originate messages. A node number is a unique number assigned by the system to a node. A terminal number is a number assigned to a specific terminal attached to a node. Terminal 0 is reserved for maintenance messaging, and the remaining terminals (1 to N) are associated with individual lines, trunks, and so on.

Terminal 0

Each node has a terminal 0 that sends and receives messages specific to the peripheral processor.

When a terminal requires a maintenance activity be performed, such as Return to Service, a message indicating that maintenance is needed is sent from terminal 0 to the CC.

When a peripheral module controller receives maintenance action commands from the MAP, such as load, busy, or test, messages are sent to terminal 0.

Therefore, if a problem is related to activities within a peripheral, monitoring the messages going to and coming from terminal 0 aids in troubleshooting.

Calculations

To determine the PM node number and terminal number composing a terminal identifier, use one of the following methods.

- Method 1. Use the CONVERT command in PMIST to determine the terminal identifier, as follows:

  CONVERT <format_id> <value>

  Refer to CONVERT on page 4-7 for the syntax for converting a LEN, a DN, or a trunk CLLI.

  Refer to Figure 2-1 for an example of the CONVERT command.

Figure 2-1xxx
CONVERT command example

```
PMIST MULTI USER:
>convert dn 6211234

NN= 0023   TN= 013C

PMIST MULTI USER:
>
```

- Method 2. Use the CI command QDN outside the PMIST subsystem with the directory number of the line to be traced as follows:

  QDN <directory number>

  Refer to Figure 2-2 for an example of using the QDN command.

Figure 2-2xxx
QDN command example

```
 CI:
 >qdn 6213010
------------------------------------------------------------
 DN:      6213010
 TYPE: SINGLE PARTY LINE
 SNPA: 613
 LINE EQUIPMENT NUMBER:    REM1 00 0 00 23
 LINE CLASS CODE:   1FR
 SIGNALLING TYPE:  DIGITONE
 LINE TREATMENT GROUP:     0
 LINE ATTRIBUTE INDEX:        0
 CARDCODE:   2X17AB   GND: N  PADGRPL STDLN  BNV: NL MNO: N
 PM NODE NUMBER        :    21
 PM TERMINAL NUMBER    :    27
 OPTIONS:
 DGT
------------------------------------------------------------
```

In the preceding example, the node number is 21 and the terminal number is 27.

- Method 3. Use the CI command QLEN outside the PMIST subsystem with the LEN as follows:

  QLEN <len>

  Refer to Figure 2-3 for an example of the QLEN command.

Figure 2-3xxx
QLEN command example

```

 CI:
 >qlen 1 0 11 1
------------------------------------------------------------
 LEN:     HOST 01 0 11 01
 TYPE: SINGLE PARTY LINE
 SNPA: 613
 DIRECTORY NUMBER:   6215111
 LINE CLASS CODE:  1FR
 SIGNALLING TYPE:  DIGITONE
 LINE ATTRIBUTE INDEX:         32
 CARDCODE  6X17    GND N PADGRP STDLN BNV NL MNO N
 OPTIONS:
 DGT
 PM NODE NUMBER     :  63
 PM TERMINAL NUMBER :  354
------------------------------------------------------------
```

In the preceding example, the node number is 63 and the terminal number is 354.

- Method 4. Use QUERYPM in the PM access level to find the node number as follows:

  POST <node type> <device number>
  QUERYPM

  Refer to Figure 2-4 for an example of the QUERYPM command.

Figure 2-4xxx
QUERYPM command example

```
PM:
>post TM8 1
POST:
>querypm
  PM TYPE TM8 PM NO.: 1 NODE NO.: 16
PM_STATUS: InSv NOTE_STATUS: OK,FALSE,CHKSUM: 018 1
PP LOAD: VALID PP EXECS: VALID FNAME: BTMIA01
PMS EQUIPPED: 30 PM INT.   :1
 Site Flr RPos  Bay_Id   Shf Description  Slot  EqPEC
 HOST  00 B01   TME 0001 04 LTC  :  000        6X02AA
```

In the preceding example, the node number is 16.

- Method 5. Use the NODENO command in PMIST to find the node number as follows:

  NODENO <node type> <device class> <device number>

  Refer to Figure 2-5 for an example of the NODENO command. For more information on the NODENO command, refer to NODENO on page 4-41.

Figure 2-5xxx
NODENO subcommand example

```
                PMIST MULTI USER:
                >nodeno tm_node tm8 1
                NODENO=16
                PMIST MULTI USER:
                >
```

Calculate the terminal number for DTC and LTC as follows:

Note: You can find the carrier number, channel number, circuit number, and similar information by posting the peripheral.

terminal number = (carrier * 32) + <channel> + 1

For TM, MTM, TM2, TM4, and similar PM, calculate the terminal number as follows:

1 + circuit number

For LM and LCM, calculate the terminal number as follows:

(drawer * 32) + <line card> + 1

- Method 6. Terminal Identifiers for DCM

  To determine the terminal number on a DCM, post the DCM at the TTP level of the MAP. Locate the carrier number and timeslot associated with the DCM.

  Refer to Figure 2-6. In the CCT column, locate the carrier number and timeslot associated with the posted DCM. The terminal number associated with the carrier and timeslot number is given in the TN column.

Figure 2-6xxx
DCM carrier and timeslot to terminal number cross-reference

CCT	TN	CCT	TN	CCT	TN	CCT	TN	CCT	TN
0-01	01	1-01	31	2-01	61	3-01	91	4-01	02
0-02	32	1-02	62	2-02	92	3-02	03	4-02	33
0-03	63	1-03	93	2-03	04	3-03	34	4-03	64
0-04	94	1-04	05	2-04	35	3-04	65	4-04	95
0-05	06	1-05	36	2-05	66	3-05	96	4-05	07
0-06	37	1-06	67	2-06	97	3-06	08	4-06	38
0-07	68	1-07	98	2-07	09	3-07	39	4-07	69
0-08	99	1-08	10	2-08	40	3-08	70	4-08	100
0-09	11	1-09	41	2-09	71	3-09	101	4-09	12
0-10	42	1-10	72	2-10	102	3-10	13	4-10	43
0-11	73	1-11	103	2-11	14	3-11	44	4-11	74
0-12	104	1-12	15	2-12	45	3-12	75	4-12	105
0-13	16	1-13	46	2-13	76	3-13	106	4-13	17
0-14	47	1-14	77	2-14	107	3-14	18	4-14	48
0-15	78	1-15	108	2-15	19	3-15	49	4-15	79
0-16	109	1-16	20	2-16	50	3-16	80	4-16	110
0-17	21	1-17	51	2-17	81	3-17	111	4-17	24
0-18	52	1-18	82	2-18	112	3-18	23	4-18	53
0-19	83	1-19	113	2-19	24	3-19	54	4-19	84
0-20	114	1-20	25	2-20	55	3-20	85	4-20	115
0-21	26	1-21	56	2-21	86	3-21	116	4-21	27
0-22	57	1-22	87	2-22	117	3-22	28	4-22	58
0-23	88	1-23	118	2-23	29	3-23	59	4-23	89
0-24	119	1-24	30	2-24	60	3-24	90	4-24	120

For example, if DCM 1 is posted at the TTP level of the MAP, and the following is displayed:

```
DCM 1 0 09
```

the carrier number is 0 and the timeslot is 09. According to the chart in Figure 2-6, 0-09 is associated with terminal number 11.

To determine the node number of the DCM, post the DCM at the PM level of the MAP and issue the command QUERYPM. The node number is contained in the output of the QUERYPM command.

Message interpreting and monitoring

This chapter describes how to monitor, interpret, and record messages sent between the CC and a PM.

Interpreting PMIST output

PMIST output is stored in the circular trace buffer or in an SFDEV file, on tape, or on a disk volume.

Intercepted messages are stored in a circular trace buffer that holds approximately 25 messages. Once the buffer reaches its capacity, the buffer will begin to overwrite itself. Also, when you exit the PMIST subsystem, the trace buffer will be cleared.

To avoid losing any messages, record the PMIST output in a SFDEV file, a tape, or a disk volume. Sending the output to a file will prevent messages from being lost and allow you to have a permanent record of monitored messages.

Incoming messages

Messages from a PM to the CC are called incoming messages or reports. The following paragraphs explain the message printout.

Following is an example of an incoming report header:

```
INCOMING 10:36:48.1 NODE TYPE=LM_NODE  ORIGINATION_MSG
NN=0014  TN= 001C MSGTAG= 3B ROUTE= 0000 ERROR= 00 LENGTH= 0A
AGENT= LEN HOST 00 0 00 27  DN 6211234
0C 00
```

The last line of the incoming report is the body of the message. The first two bytes of the message body are the message type. The remaining bytes contain information about what has occurred at the PM, such as the digits dialed.

Refer to Table 3-1 for descriptions of incoming message fields.

Table 3-1
Incoming message fields

Field	Description
NODE TYPE	is the type of PM sending the report.
message type	follows the node type. An example is an origination message. For more information on message types, see Table 3-2.
NN	is the node number in hexadecimal of the PM sending the report.
TN	is the terminal number in hexadecimal of the circuit sending the report.
MSGTAG	is the message tag, which is the first byte of the incoming message.
ROUTE	indicates which route the message took through the switch.
ERROR	indicates an error in transmission.
LENGTH	is the total number of bytes in the message. Length is indicated in hexadecimal.
AGENT	is the call processing agent, such as a LEN or a circuit.
DN	is the directory number of the calling party.
LBYTE	is the last byte of the incoming message, indicating the unit number from which the message was received. LBYTE is not always displayed.

End

Incoming messages on PMIST output contain the type of message being sent from a PM.

The message types in Table 3-2 are the basic message types that may be reported to the CC by a PM during the normal setup and/or completion of a call.

Table 3-2xx
Message types

Message types	Explanation	
Abandon_Msg	indicates the calling party has gone on-hook before all required digits have been collected and before a permanent signal, partial dial, or fast interdigit time-out has occurred. This message contains received digits.	
ANI_MSG	reports results of Automatic Number Identification (ANI) on a party line.	
Answer_msg	indicates hardware answer has been detected.	
Atd_Result_Msg	is reported by an Audio Tone Detector (ATD). It contains information describing the events detected in response to a particular request by the CC, such as dialtone detection and voice detection. The responses that can be reported by the ATD within an Atd_Result_Msg are as follows:	
	ATD_DIAL	indicates dialtone was detected.
	DIALTONE_NOT_DET	indicates no dialtone was detected.
	TOO_MANY_RINGS	indicates the telephone rang too often.
	ATD_BUSY	indicates a busy tone was detected.
	ATD_REORDER	indicates a reorder tone was detected.
	HI_DRY_TIMEOUT	indicates dead silence was detected.
	DEFAULT_ANS	indicates a possible answer.
	ATD_VOICE	indicates a voice was detected.
	NO_CSM_TRANS	indicates CSM transmission trouble.
Call_Abandoned_Msg	indicates a line was exited during digit collection.	

-continued-

Table 3-2xx
Message types (continued)

Message types	Explanation
`Call_Failure_Msg`	indicates that an invalid port-side digital signaling condition was received by the PM when the trunk was not in the idle state or that an unrecoverable error was located by the PM execs during the processing of a call. This message contains information that describes the type of error that occurred as well as the contents of some of the terminal storage areas used for error handling and debugging purposes. The call failure error types are as follows: `1` indicates that an outpulsing problem occurred. An example of this type of error is when an outgoing trunk uses wink signaling to transmit an off-hook signal but does not receive a wink signal from the far end within a specified amount of time. This would be a “no start dial” condition. `2, 3` indicates an invalid signaling condition was received. An example of this type of error is when a trunk using “AB” bit signaling receives an unexpected “AB” condition.
`Channel_Blocking_Msg`	indicates a call was blocked in an XPM because of insufficient hardware resources.
`Clear_Back_Msg`	indicates the called party went on-hook. This message indicates whether the call was actually answered and the duration of the call in 10 ms ticks. If the call was not answered, the time reflects duration of the call since call origination.
`Clear_Forward_Msg`	indicates the calling party went on-hook. This message indicates whether the call was actually answered and the duration of the call in 10 ms ticks. If the call was not answered, this time reflects the duration of time since call origination.

-continued-

Table 3-2xx
Message types (continued)

Message types	Explanation
Coin_Msg	reports the results of a coin control function on a coin line.
Conf_Available	indicates a conference circuit is available.
Confusion_Msg	is reported by a PM audit when the audit has found a trunk that appears to have been neglected for a given period of time.
CPOS_Available	indicates a Centralized Automated Message Accounting (CAMA) position is available.
CP_wakeup_msg	reports time-out from the CP wakeup utility.
Dgt_Reception_Err_Msg	indicates an error was detected during DTMF digit collection.
Digits_Msg	contains the digits received by the PM since the last request for digits was made by the CC.
Digits_Sent_Msg	indicates all digits that were to be outpulsed have been outpulsed.
DM_Report_Msg	reports key strokes from Traffic Operator Position System (TOPS), Auxiliary Operator Service System (AOSS), and Integrated Business Network (IBN).
DP_Reception_Err_Msg	indicates an error was detected during DP digit collection.
Exit_Msg	indicates a line was exited.
Feature_Msg	requests invocation of a Feature Processing Environment (FPE) feature.
Flash_Msg	indicates a terminal flashed.
Glare_Msg	indicates a glare was detected on a trunk.
Integ_Found_Msg	indicates a terminal found integrity.
Integ_Lost_Msg	indicates an established connection has lost integrity.
Intra_Blocking_Msg	indicates a XPM could not intraswitch and rerouted through network modules.
Last_Digits_Msg	indicates the digit reception process was idled and no more digits will be collected until a digit reception process is started again. This message is reported when a permanent signal time-out occurs, when a partial dial time-out occurs, or when all the digits that are required for the given digit stream have been collected. The message contains received digits.

-continued-

Table 3-2xx
Message types (continued)

Message types	Explanation
Lockout_Msg	indicates that an invalid port-side digit signaling condition was received by the PM when the trunk was in the idle state. This message also indicates that an incoming idle condition was not received within a specified amount of time when attempting to place a trunk back in the idle state.
Operator_Control_Msg	indicates winks were detected on an operator trunk.
Orig_Digits_Msg	indicates an incoming seizure has been recognized and contains the initial set of digits received.
Orig_Key_Msg	indicates a business set is originating a call on a non-DN key.
Origination_Msg	indicates an incoming seizure has been recognized on a particular trunk.
Preempt_Clear_Msg	indicates a trunk exited due to the other end preempt.
Preempt_Internal_Msg	is sent from preempting call to preempted call.
Preempt_Reuse_Msg	releases a trunk for use by a new call.
Release_Call_Msg	indicates an invalid port-side digit signaling condition was received by the PM when the trunk was in the idle state. This message also indicates that an incoming idle condition was not received within a specified amount of time when attempting to place a trunk back in the idle state.
RCVR_Available	indicates a receiver is available.
RCVR_Error_Msg	indicates an error was detected during digit reception.
Release_Call_Msg	indicates a terminal will be forced to release.
Remote_Busy_Msg	indicates a line will be forced to release.
Ringing_trouble_Msg	indicates an error was detected during the physical ringing of a line.
Routing_Msg	causes the setup processor to invoke the router.
Seize_msg	indicates an outgoing seizure has been completed on a trunk.
SVCT_Msg	indicates the sender is available.
Treatment_Msg	causes the setup processor to apply treatment.

-continued-

Table 3-2xx
Message types

Message types	Explanation
UTR_Denied	indicates an XPM could not obtain a receiver for digit collection.
Wink_Msg	indicates a wink was detected on a CAMA trunk.

End

Outgoing messages

Outgoing messages are instructions sent from the CC to a PM. Outgoing message headers are similar to those of incoming messages, except the message type does not appear in the first line of the message header and the message body format is different. In addition, a list of Operation Codes (opcodes) follow the message body. Opcodes identify the particular operation to be performed and indicate to the CPU what to do and how to interpret any parameters that may follow.

The first column of the list is the address, or offset, of the opcode within the message body. In this example, opcode 80 is at offset 00 of the outgoing message body. Opcode 2D is at offset 02 and so on.

The second column is the primitive opcode, and the symbolic name of the opcode is in the third column. The symbolic name may be followed by the parameters for the opcode. The basic list of the primitives that appear in PMIST output is in Chapter 7.

In the outgoing message body the opcodes are followed by their parameters. For instance, opcode 80 requires that 1F be copied onto the parameter stack. Therefore, 80 is at position 00 of the message body and 1F is at position 01.

Following is an example of an outgoing message:

```
OUTGOING 10:36:48.1 NODE TYPE= LM_NODE
NN= 0014 TN= 001C MSGTAG= 00 ROUTE= 068A ERROR= 00 LENGTH= 0E
AGENT= LEN HOST 00 0 00 27  DN 6211234
80 1F 2D 65 38 AE

00      80    STKI1      1F
02      2D    ASSCHN
03      65    STPIDSCAN
04      38    CALLXEC    AE
```

Sample session

The following is the output generated by a line to equal access trunk call. The output is broken up with comments clarifying the significance of each message.

```
INCOMING 10:36:48.1 NODE TYPE=LM_NODE ORIGINATION_MSG
NN= 0014 TN= 001C MSGTAG= 00 ROUTE= 0000 ERROR= 00 LENGTH= 0A
AGENT= LEN HOST 00 0 00 27 DN 6211234
0C 00
```

A Line Module (LM) signals the CC that an off-hook condition occurred by sending an origination message.

```
OUTGOING 10:36:48.1 NODE TYPE= LM_NODE
NN= 0014 TN= 001C MSGTAG= 00 ROUTE= 068A ERROR= 00 LENGTH= 0E
AGENT= LEN HOST 00 0 00 27  DN 6211234
80 1F 2D 65 38 AE

00 80STKI1  1F
02 2DASSCH
03 65STPIDSCAN
04 38CALLXECAE
```

The CC responds by instructing the PM to associate a channel with the terminal (ASSCHN), stop scanning to determine whether the line is idle (STPIDSCAN), and to invoke an exec (CALLXEC) with exec_id AE. (Refer to the glossary for a definition of an exec.)

```
INCOMING 10:36:55.2 NODE TYPE= LM_NODE  DIGITS_MSG
NN= 0014 TN= 001C MSGTAG= 3B ROUTE= 0000 ERROR= 00 LENGTH= 13
AGENT= LEN HOST 00 0 00 27  DN 6211234
16 00 81 88 00 00 00 00 00 40 40
```

Next, the PM sends a digits message that contains the digits dialed by the calling party. The first incoming message contains the first four numbers dialed by the calling party. This information is found in the message body.

```
16 00 81 88 00 00 00 00 00 40 40
```

The message type (16 00) indicates that this message is a digits message. The next four bytes (81 88) are the first four digits dialed by the calling party. The dialed digits are read according to the following formula: BA DC. Read the formula alphabetically.

Therefore, the first four dialed digits are 1888. At the end of the message, 40 40 appears. In this example, the first 40 is the number count for both incoming digits messages, and the second 40 is the number of digits reported for both messages.

At this point, the CC seizes an idle trunk before receiving the last four digits. The CC action causes a break in digit reception.

```
INCOMING 10:36:58.8 NODE TYPE= LM_NODE  DIGITS_MSG
NN= 0014 TN= 001C MSGTAG= 3B ROUTE= 0000 ERROR= 00 LENGTH= 13
AGENT= LEN HOST 00 0 00 27  DN 6211234
16 00 21 43 00 00 00 00 00 40 40
```

In the second incoming message, the message body contains the last of the digits dialed (21 43). The dialed digits are read according to the following formula: BA DC. Read the formula alphabetically. Therefore, the last four digits are 1234.

```
OUTGOING  10:36:58.9 NODE TYPE= TM_NODE
NN= 0025  TN= 000C MSGTAG= 00 ROUTE= 168A ERROR= 00 LENGTH=1B
AGENT= CKT        OGEAMCI    1
81 3A 0C 2D 1A 84 1C 93 33 02 FF E0 13 E0 14 E0 16 E1 07

00 81STKI23A 0C
03 2DASSCHN
04 1AXMNTG
05 84STKI51C 93 33 02 FF
0B E0POPV1  13
0D E0POPV1  14
0F E0POPV1  16
11 E1POPV2  07
```

The CC sends three messages following the digits messages. The first message instructs the Trunk Module (TM) to associate a channel with the terminating trunk.

```
OUTGOING  10:36:58.9  NODE TYPE= LM_NODE
NN= 0014 TN=001C MSGTAG= 00 ROUTE= 0602 ERROR= 00 LENGTH= 31
AGENT= LEN HOST 00 0 00 27  DN 6211234
82 80 90 3A E0 1C 90 1B 47 11 02 2A 85 40 3A 00 0C 44 0C E0 1A
E0 18 6B 38 CB 80 D9 E0 08 80 00 E0 1E 80 04 90 1E 48 E0 1E

00 82STKI380 80 3A
04 E0POPV1  1C
```

```
06 90STKI1  1B
08 47ANDPARMS
09 11*IFNE  02
0B 2ASWCHPCM
0C 85STKI640 3A 00 0C 44 0C
13 E0POPV1  1A
15 E0POPV1  18
17 6BORGSUSUP
18 38CALLXECCB
1A 80STKI1  D9
1C E0POPV1  08
1E 80STKI1  00
20 E0POPV1  1E
22 80STKI1  04
24 90STKI1  1E
26 48ORPARMS
27 E0 POPV1 1E
```

The second message instructs the PM to select Pulse Code Modulation (PCM) input from the other network plane and to start a terminal process to perform origination set up and supervision.

```
OUTGOING  10:36:58.9 NODE_TYPE= TM_NODE
NN= 0025 TN= 000C MSGTAG= 00 ROUTE= 168A ERROR= 00 LENGTH= 1B
AGENT= CKT       OGEAMCI   1
84 80 00 3A 80 80 90 1B 47 11 02 2A E0 1C 2F 2F 90 09 26

00 84STKI580 00 3A 80 80
06 90STKV1  1B
08 47ANDPARMS
09 11IFNE 02
0B 2ASWCHPCM
0C E0POPV1  1C
0E 2FSETPAD
0F 2FSETPAD
10 90STKV1  09
12 26NVKXEC
```

The third message instructs the TM to connect PCM and apply appropriate padding to the call.

```
INCOMING 10:37:00.4 NODE TYPE= TM_NODE EQUAL_ACCESS_WINK_MSG
NN= 0025 TN= 000C MSGTAG= 3B ROUTE= 4080 ERROR= 00 LENGTH= 0B
AGENT= CKT       OGEAMCI    1
```

```
FE 00 02
```

The equal access wink message indicates that the call can proceed.

```
OUTGOING 10:37:00.4 NODE TYPE= TM_NODE
NN= 0025 TN= 000C MSGTAG= 00 ROUTE= 168A ERROR= 00 LENGTH= 37
AGENT= CKT       OGEAMCI    1
87 AD 6A 31 26 11 32 F4 E0 E7 0A 87 07 E4 1C 5E 2B 0C 00 00 82
00 A0 92 E0 14 59 4C 12 E4 00 E0 17 26 80 01 90 03 0E 80 00 E0
16 80 0C E0 06

00 87STKI8AD 6A 31 26 11 32 F4 E0
09 E7POPV8  0A
0B 87STKI807 E4 1C 5E 2B 0C 00 00
14 82STKI300 A0 92
18 E0POPV1  14
1A 59PATHCON
1B 4CPOPNO  12
1D E4POPV5  00
1F E0POPV1  17
21 26NVKXEC
22 80STKI1  01
24 90STKV1  03
26 0ESTIME
27 80STKI1  00
29 E0POPV1  16
2B 80STKI1  0C
2D E0POPV1  06
```

The CC response to the wink message is to outpulse the calling digits to the TM for billing purposes. The calling digits are at the beginning of the message. In the following sequence, `87 AD 6A 31 26 11 32 F4 E0`, the D represents the KP signal, and numbers that follow are the calling digits. The A is read as a 0. The F represents the start signal and indicates the end of the calling digits, which are read according to the following formula. Read the formula alphabetically: `BA DC FE HG JI`. Therefore, the calling digits are 6136211234.

```
INCOMING 10:37:02.6 NODE TYPE= TM_NODE DIGITS_SENT_MSG
NN= 0025 TN= 000C MSGTAG= 3B ROUTE= 4080 ERROR= 00 LENGTH= 0A
AGENT= CKT       OGEAMCI    1
1C 00
```

The digits sent message informs the CC that the digits the PM was instructed to outpulse have been outpulsed.

```
OUTGOING 10:37:02.6 NODE TYPE= TM_NODE
NN= 0025 TN= 000C MSGTAG= 00 ROUTE= 1602 ERROR= 00 LENGTH= 33
AGENT= CKT       OGEAMCI    1
87 8D 88 21 43 0F 00 00 90 E7 0A 82 32 10 02 E0 14 E0 14 E0 16
E0 07 38 00 81 5F A7 E1 14 80 A0 59 80 32 E0 13 38 52 83 01 80
80 10 58

00 87STKI88D 88 21 43 0F 00 00 90
09 E7POPV8  0A
0B 82STKI332 10 02
0F E0POPV1  14
11 E0POPV1  16
13 E0POPV1  07
15 38CALLXEC00
17 81STKI25F A7
1A E1POPV2  14
1C 80STKI1  A0
1E 59PATHCON
1F 80STKI1  32
21 E0POPV1  13
23 38CALLXEC   52
25 83STKI401 80 80 10
2A 58CSMMON
```

Next, the CC outpulses the called digits to the TM. The calling digits are at the beginning of the message. In the following sequence, 8D88 21 43 0F 00, the D indicates that the numbers that follow are the called digits. The F indicates the end of the called digits. The called digits are read according to the following formula: BA D C FE HG JI. Therefore, the called digits are 8881234.

```
INCOMING 10:37:08.1 NODE TYPE= LM_NODE ANSWER_MSG
NN= 0014 TN= 001C MSGTAG= 3B ROUTE= 0000 ERROR= 00 LENGTH= 0E
AGENT= LEN HOST 00 0 00 27   DN 6211234
19 00 96 03 00 00
```

The answer message from the LM indicates the called party has gone off-hook.

```
OUTGOING 10:37:08.1 NODE TYPE= TM_NODE
NN= 0025 TN= 000C MSGTAG= 00 ROUTE= 1602 ERROR= 00 LENGTH= 0C
```

```
AGENT= CKT       OGEAMCI    1
80 00 E0 08

00 80STKI1  00
02 E0POPV1  08
```

The CC then sends information to the TM to nil the information located at offset 8.

```
INCOMING 10:37:13.0 NODE TYPE= LM_NODE  EXIT_MSG
NN= 0014 TN= 000C MSGTAG= 3B ROUTE= 0000 ERROR= 00 LENGTH= 0F
AGENT= LEN HOST 00 0 00 27   DN 6211234
75 00 8D EE 01 00 00
```

Finally, the exit message indicates the calling party has gone on-hook.

```
OUTGOING 10:37:13.0 NODE TYPE= LM_NODE
NN= 0014 TN= 000c MSGTAG= 00 ROUTE= 068A ERROR= 00 LENGTH= 11
AGENT= LEN HOST 00 0 00 27   DN 6211234
3A 3A 82 03 00 02 52 2E 64

00 3ASVOFSET
01 3ASVOFSET
02 82STKI303 00 02
06 52SETSDG
07 2EDISCHN
08 64STAIDSCAN
```

The CC instructs the LM to disassociate the terminal from a channel and to start scanning the idle line for an off-hook condition.

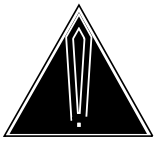
Monitoring a call

To monitor a call, perform the following steps:

1 Invoke PMIST by entering `PMIST`.

Note: Invoke XPMIST by entering the following command: `XPMIST`.

2 Ensure that only the terminal identifiers you specify will be monitored by entering `SELECT ON`.



CAUTION

Enter command select on each time PMIST is accessed. If this subcommand is not entered, every message in the office will be intercepted, which can cause the switch to crash.

3 Ensure that I/O messages will be monitored by entering `INTERCEPT BOTH MON`.

CAUTION

Enter `INTERCEPT BOTH MON` each time PMIST is accessed. If this subcommand is not entered, messages will not be allowed to reach their destination, possibly causing the switch to crash.

4 Enter the terminal identifiers of the PM that will be monitored by entering `INCLUDE <node number> <terminal number>`.

Note: To calculate the node and terminal numbers, see Chapter 2.

5 Verify that the data was entered accurately by typing `PMIST`.

The current status of PMIST appears at the MAP.

6 Create a file to store intercepted messages by entering `RECORD OPEN`.

A prompt for a device name and filename will appear. Enter `<filetype> <new filename>`.

An example of the filetype is SFDEV. Each call monitored with PMIST requires a unique filename.

7 Clear the circular buffer of all previously intercepted messages by entering `CMSGLIST`.

8 Be sure that the circular buffer is empty by entering `DMSGLIST`.

If the buffer is empty, the only message that will be displayed is "END OF INTERCEPTED I/O MESSAGES." If there are other messages, enter `CMSGLIST`. Then, enter `DMSGLIST` again to ensure that the buffer is cleared.

9 Make the test call.

Note: The circular buffer can hold messages from only one completed call; therefore, do not try to make more than one call or the buffer will overwrite itself.

10 When the call is complete, close the record file by entering `RECORD CLOSE`.

11 To make a hard copy of the intercepted messages, enter `SEND <printer name>`.

12 Display the intercepted messages by entering `DISPLAY <filename>`.

Note: The file created by the RECORD subcommand cannot be displayed with the CI command PRINT.

13 To leave the PMIST environment, enter `QUIT`.

The QUIT subcommand removes any terminal identifiers entered with the INTERCEPT or INCLUDE commands during the PMIST session.

A sample line to line message trace

shows an example of setting up a line to line call following the steps previously outlined in this section.

Figure 3-1xxx
Sample PMIST session (Part 1 of 2)

```
CI:
>qdn 6213010
------------------------------------------------------------
DN:       6213010
TYPE: SINGLE PARTY LINE
SNPA: 613
LINE EQUIPMENT NUMBER:      REM1 00 0 00 23
LINE CLASS CODE:   1FR
SIGNALING TYPE:   DIGITONE
LINE TREATMENT GROUP:    0
LINE ATTRIBUTE INDEX:      0
CARDCODE  2X17AB     GND: N   PADGRP: STDLN  BNV: NL MNO: N
PM NODE NUMBER     :    21
PM TERMINAL NUMBER :    24
OPTIONS:
DGT
------------------------------------------------------------
CI:
>qdn 6213050
------------------------------------------------------------
DN:       6213050
TYPE: SINGLE PARTY LINE
SNPA: 613
LINE EQUIPMENT NUMBER:      REM1 00 0 00 26
LINE CLASS CODE:   1FR
SIGNALING TYPE:   DIGITONE
LINE TREATMENT GROUP:    0
LINE ATTRIBUTE INDEX:      0
CARDCODE  2X17AB     GND: N   PADGRP: STDLN  BNV: NL MNO: N
PM NODE NUMBER     :    21
PM TERMINAL NUMBER :    27
OPTIONS:
DGT
------------------------------------------------------------
```

Figure 3-1xxx
Sample PMIST Session (Part 2 of 2)

```
CI:
>pmist

PMIST MULTI USER:
>select on

PMIST MULTI USER:
>intercept both mon

PMIST MULTI USER:
>include 21 24
PMIST MULTI USER:
>include 21 27
PMIST MULTI USER:
>record open
ENTER DEVICE NAME AND NEW FILE NAME:
>sfdev newfile
PMIST MULTI USER:
>cmsglist
PMIST MULTI USER:
>dmsglist
END OF INTERCEPTED I/O MESSAGES.
PMIST MULTI USER:
>pmist
INTERCEPT STATUS:      INCOMING MONITOR,   OUTGOING MONITOR
NO NODE TYPES SPECIFIED INTERCEPT-IMMUNE
SELECTIVE INTERCEPT ENABLED;  2 TERMINAL IDENTIFIERS SPECIFIED
      NN=  0015    TN=  0036
      NN=  0015    TN=  0039
I/O MESSAGE RECORDING DISABLED
RESTART IS OFF
PMIST MULTI USER:

%Call is made
>record close
PMIST MULTI USER:
>send prt1
PMIST MULTI USER:
>display newfile
```

PMIST level commands

Use the PMIST level of the MAP to record incoming and outgoing messages to help determine whether the CC is responding properly to PM signals and whether the PM is processing CC instructions correctly.

Accessing the PMIST level

To access the PMIST level, enter the following from the CI level:

pmist ↵

PMIST commands

The commands available at the PMIST MAP level are described in this chapter and arranged in alphabetical order. The page number for each command is listed in the following table.

PMIST commands

Command		Page
changebuf	PMIST	4-3
cmsglist	PMIST	4-5
convert	PMIST	4-7
devicename	PMIST	4-9
disconn	PMIST	4-11
display	PMIST	4-13
dmsglist	PMIST	4-15
extract	PMIST	4-17
help	PMIST	4-21
immune	PMIST	4-23
include	PMIST	4-25
insert	PMIST	4-27

-continued-

PMIST commands (continued)

Command		Page
inserbi	PMIST	4-29
insertduplex	PMIST	4-31
insmg	PMIST	4-33
intercept	PMIST	4-37
nodeno	PMIST	4-41
pmist	PMIST	4-43
quit	PMIST	4-47
reconn	PMIST	4-49
record	PMIST	4-51
remove	PMIST	4-53
restart	PMIST	4-55
select	PMIST	4-57
terminate	PMIST	4-59
tmsglist	PMIST	4-61
verify	PMIST	4-63

-end-

changebuf

Function

Use the changebuf command to change the size of the circular buffer.

changebuf command parameters and variables

Command	Parameters and variables
changebuf	*bufsize*

Parameters and variables	Description
bufsize	Specifies the new buffer size. Valid range is 1 through 246.

Qualifications

If no parameter is entered, the current buffer size is displayed.

WARNING

The changebuf command clears the old buffer.

By issuing this command, the old message buffer is cleared. The new buffer size erases the old buffer.

Examples

None available

Responses

The following table provides an explanation of the responses to the changebuf command.

Responses for the changebuf command

MAP output	Meaning and action
`THE CURRENT BUFFER SIZE IS`	**Meaning:** No parameter was entered with the changebuf command. **Action:** None

-continued-

changebuf (end)

Responses for the changebuf command (continued)

MAP output	Meaning and action
NO ACTION TAKEN	**Meaning:** The no option was chosen at the prompt. **Action:** None
-end-	

cmsglist

Function

Use the cmsglist command to clear the list of intercepted I/O messages.

cmsglist command parameters and variables

Command	Parameters and variables
cmsglist	There are no parameters or variables.

Qualifications

If you access the PMIST subsystem by entering `READ PUPI`, you can enter `CL`, which is the shortened form of CMSGLIST. Refer to Chapter 6 for more information on the PUPI level.

Examples

None available

Responses

None available

convert

Function

Use the convert command to calculate the associated node and terminal number for a given Directory number (DN), Line Equipment Number (LEN), or trunk CLLI.

convert command parameters and variables

Command	Parameters and variables		
convert	dn	*dn*	
	len	*len*	
	trk	*clli*	*ext_trk_name*

Parameters and variables	Description
clli	Specifies the trunk CLLI.
dn	Indicates the specified DN will be converted to a TID.
dn	Indicates the directory number. Only 7-digit DNs are supported.
ext_trk_name	Specifies the external trunk name.
len	Indicates a LEN to be converted to a TID.
len	Specifies the line equipment number.
trk	Specifies a trunk CLLI to be converted to a TID.

Qualifications

Once the convert command is processed, you can enter the include or the remove commands without any parameters. The results of the last convert command will be included in or removed from the TIDs to be monitored.

The convert command converts CLLIs associated with trunks whose CP_SELECTOR type is TRUNK_CPS. Facilities outside of PMIST must be used to calculate the TID for trunks with a CP_SELECTOR type that is not equal to TRUNK_CPS.

Examples

None available

convert (end)

Responses

The following table provides explanations of the responses to the convert command.

Responses for the convert command

MAP output	Meaning and action
`INVALID DIRECTORY NUMBER`	**Meaning:** The DN specified with the DN parameter has not been assigned. **Action:** Verify the DN to be converted and reenter the data.
`DNS UNAVAILABLE`	**Meaning:** The switch on whcih the DN terminates is not a local office. **Action:** Verify the DN to be converted and reenter the data.
`CLLI-EXTRKNM IS NOT A TRUNK`	**Meaning:** The CLLI specified with the TRK parameter is not a valid trunk name. **Action:** Verify the CLLI and external trunk name and reenter data.

devicename

Function

Use the devicename command to convert the node number to the device name.

devicename command parameters and variables

Command	Parameters and variables
devicename	*node_no*

Parameters and variables	Description
node_no	Specifies the node number. Valid range is 0 through 4095.

Qualifications

None

Examples

None available

Responses

The following table provides an explanation of the response to the devicename command.

Responses for the devicename command

MAP output	Meaning and action
`NODE UNDEFINED`	**Meaning:** The specified node is undefined. **Action:** Verify node number and reenter data.

disconn

Function

Use the disconn command to disconnect the user while the system is still monitoring and recording the session.

disconn command parameters and variables

Command	Parameters and variables
disconn	*userid*

Parameters and variables	Description
userid	Specifies the name of the PMIST user to be disconnected.

Qualifications

The disconn command is qualified by the following exceptions, restrictions, and limitations:

- Must have access to tool supervisor (TOOLSUP) which is password controlled.
- Must be in the PMIST environment.

Examples

The following table provides examples of the disconn command.

Examples of the disconn command

Example	Task, response, and explanation	
disconn ↵		
	Task:	Disconnect the PMIST user with the session continuing to monitor and record messaging for this session.
	Response:	`DISCONN SESSION STARTED` `PMIST TERMINATES.`
	Explanation:	The user disconnected his own CI process from PMIST and quit the environment while the montioring and recording of messaging is continuing to be performed by the DMS system.

-continued-

disconn (end)

Examples of the disconn command

Example	Task, response, and explanation

disconn ↵*userid*
where

userid is the name of the user running a session in PMIST.

Task: Disconnet the PMIST specified PMIST user with the session continuing to monitor and record messages.

Response:
```
DISCONN SESSION STARTED
PMIST TERMINATES.
```

Explanation: The user disconnected a specified CI process from the PMIST environment while the montioring and recording of messages is continuing to be performed by the DMS system.

-end-

Responses

The following table provides an explanation of the response to the disconn command.

Responses for the disconn command

MAP output	Meaning and action

```
NO RECORDING IN PROGRESS. CHECK STATUS.
```

Meaning: The RECORD ON command was not executed before attempting to disconnect a PMIST user.

Action: Execute the RECORD ON command to begin recording before the disconnect session is started.

display

Function

Use the display command to display the message sequence stored in the message file created with the record command.

display command parameters and variables

Command	Parameters and variables
display	*filename*

Parameters and variables	Description
filename	Specifies the name of the file to be displayed.

Qualifications

None

Examples

None available

Responses

The following table provides explanations of the responses to the display command.

Responses for the display command

MAP output	Meaning and action
`FILE SPECIFIED NOT A MESSAGE FILE`	**Meaning:** The file name entered with the display command was not an I/O message file name. **Action:** Verify filename to be entered, check spelling, and reenter command and filename.
`INVALID FILE`	**Meaning:** An invalid file name was entered. **Action:** Verify filename to be entered, check spelling, and reenter the command and filename.

dmsglist

Function

Use the dmsglist command to display the intercepted I/O messages once they have been logged in the message list contained in the circular buffer.

dmsglist command parameters and variables

Command	Parameters and variables
dmsglist	*number*

Parameters and variables	Description
number	Specifies the number of the intercepted I/O messages to be displayed. Valid range is 1 through 246.

Qualifications

If no parameter is entered, all messages in the message list are displayed.

If you access PMIST with the read pupi command, you can enter `D`, which is the shortened form of DMSGLIST. Refer to Chapter 6 for more information on the PUPI level.

Examples

None available

Responses

None available

extract

Function

Use the extract command to display the I/O messages that occurred between the user-specified start and stop times. You can use this command to look at a specific sequence of messages within a large message file.

extract command parameters and variables

Command	Parameters and variables				
extract	open	*filename*			
	close				
	from	[*hours*	*minutes*	*seconds*	*tenths*]
	to				
	for	*no_of_msgs*			
	tid	*node_no*	*terminal_no*		

Parameters and variables	Description
close	Closes a message file that has been opened
filename	Specifies the name of the message file to be opened
for	Indicates that only a limited number of messages should be displayed. For example, if you want to look at only six messages, you would enter EXTRACT FOR 6. The number of messages is from 0 to 32767.
from	Indicates that all messages recorded before the time specified with this parameter will not be displayed. The default is 00:00:00.0. For example, if you performed a PMIST trace that began at 18:54:31.5 and ended at 19:32:32.6, you can view the messages that occurred FROM 19:00:00 onward.
hours	Specifies the time in hours that the extracted message begins or eneds. The valid entries are 0 through 99.
minutes	Specifies the time in minutes that the extracted message begins or ends. The minutes parameters are from 0 to 59.
no_of_msgs	Specifies the number of messages to be displayed. The number of messages range from 0 to 32,767.
node_no	Identifies the node whose messages are extracted. The node numbers are from 0 to 4095.

-continued-

extract (continued)

extract command parameters and variables (continued)

Parameters and variables	Description
open	Opens a message file that already exists
seconds	Specifies the time in seconds that the extracted message beings or ends. The seconds parameters are from 0 to 59.
tenths	Specifies the time in tenths of seconds that the extracted message begins or ends. The tenths of a second parameters are from 0 to 9.
terminal_no	Identifies the terminal whose messages are extracted. The terminal numbers are from 0 to 4095.
tid	Specifies the terminal identifier. This parameter allows you to view those message that were sent to and from one particular terminal. Only messages to and from this terminal are extracted. The valid range for node is 0 through 4095 and the valid range for terminal is 0 to 6095.
to	Indicates that all messages recorded after the time specified with this parameter will not be displayed. For example, if you performed a PMIST trace that began at 18:54:31.5 and ended at 19:32:32.6, you can view the messages that occurred FROM 18:54:31.5 TO 19:10:54.3. If no time is entered, the default is 99, 59, 59, 9.

-end-

Qualifications

None

Examples

None available

Responses

The following table provides explanations of the responses to the extract command.

extract (end)

Responses for the extract command	
MAP output	**Meaning and action**
`FILE SPECIFIED NOT A MESSAGE FILE`	
	Meaning: The specified file is not a message file. **Action:** Verify message filename, check filename spelling, and reenter filename.
`NO EXTRACT FILE OPENED`	
	Meaning: No message file has been open. **Action:** Retry the command using the open parameter.
`END OF FILE ENCOUNTERED`	
	Meaning: Self-explanatory. **Action:** None
`INVALID FILE`	
	Meaning: Self-expanatory **Action:** Verify filename and reenter the command.

help

Function

Use the help command to display a list of the PMIST commands. Instructions are included on how to obtain more detailed information on each command.

help command parameters and variables

Command	Parameters and variables
help	There are no parameters or variables.

Qualifications

None

Examples

The following table provides an example of the help command.

Examples of the help command

Example	Task, response, and explanation
help ↵	

Task: Display the list of PMIST commands.

Response: Available PMIST subcommands are:

```
PMIST       QUIT        TERMINATE
INSERT      INSERTBI    INSERTDUPLEX
INSMSG      INTERCEPT   IMMUNE
SELECT      INCLUDE     REMOVE
RECORD      DISPLAY     EXTRACT
VERIFY      DMSGLIST    CMSGLIST
TMSGLIST    NODENO      DEVICENAME
RESTART     CHANGEBUF   HELP
CONVERT
Use PMIST SUBC for the format of all subcommands
Use Q for the format of an individual subcommand
```

Explanation: The list is displayed.

Responses

None available

immune

Function

Use the immune command to specify the node type whose I/O messages will not be subject to interception. A large proportion of the messages passing through the I/O system can be subject to immunity based on the source or destination node types.

immune command parameters and variables

Command	Parameters and variables
immune	add delete [*nodetype*]

Parameters and variables	Description
add	Allows nodetypes to be specified as immune.
delete	Removes nodetypes from intercept immunity.
nodetype	Indicates the node that is intercept immune.

Qualifications

If no parameters are entered, the current intercept-immune node types are displayed. The intercept command also reenables interception of a nodetype that had been made immune.

The immune command is used when the select command is off.

Examples

None available

Responses

The following table provides explanations of the responses to the immune command.

immune (end)

Responses for the immune command	
MAP output	**Meaning and action**
`ERROR - - NO IMMUNE NODES LIST IN MULTI-USER PART`	
	Meaning: The immune command can only be used in the single user environment.
	Action: Message other users if you need to use the immune command.
`IMMUNE NODE TYPES LIST FULL`	
	Meaning: No new nodes can be added to the list of immune node types.
	Action:

include

Function

Use the include command to designate one or more terminal identifiers for selective intercept or monitoring.

include command parameters and variables

Command	Parameters and variables
include	*node_no* *terminal_no* mta

Parameters and variables	Description
mta	Specifies the Message Transport Address option, which adds 4096 to the node number being entered with the include command. This allows messages that use the Message Transport System protocol to be monitored.
node_no	Specifies the node number to be included. The node number is from 0 to 4095.
terminal_no	Specifies the terminal number to be included. The terminal number is from 0 to 4095.

Qualifications

It is important to have the select command on when using the include command.

More than one node number and terminal number pair can be entered with the include command. For example:

```
INCLUDE 15 36 15 39
```

If the include command is entered with no parameters, the result of the last convert command will be included in the list of TIDs to be monitored.

Examples

None available

Responses

The following table provides explanations of the responses to the include command.

include (end)

Responses for the include command

MAP output	Meaning and action
`WARNING: MORE THAN 32 SELECTED TIDS MAY CAUSE DEGRADATION.`	**Meaning:** If more than 32 terminals are specified, total system degradation is possible. **Action:** You may decide whether or not to continue.
`INVALID TERMINAL IDENTIFIER` `TID BEING PROCESSED WAS`	**Meaning:** The system does not recognize the node and terminal numbers. **Action:** Verify terminal identifier and reenter data.
`ERROR - - MAXIMUM NUMBER OF NODES INCLUDED BY ALL USERS`	**Meaning:** No more nodes can be entered by any user. **Action:** None.

insert

Function

Use the insert command to inject a user-specified incoming or outgoing message into the I/O message stream. PMIST supplies the message header.

insert command parameters and variables

Command	Parameters and variables		
insert	in out	*node_no* *terminal_no*	[*data_string*]

Parameters and variables	Description
data_string	Specifies bytes of data from 0 to 255. Up to 247 bytes of data can be entered. Data can be entered in either decimal or hexadecimal.
in	Specifies an incoming message will be inserted.
node_no	Specifies the node number for the terminal from which the message originates. Valid range is 0 through 4095.
out	Specifies that an outgoing message will be inserted.
terminal_no	Specifies the terminal number from which the message is intended. Valid range is 0 through 4095.

Qualifications

If your message is longer than the width of the terminal screen, add a plus sign (+) to the end of the line.

CAUTION

INSERTing incorrect data into the I/O system can cause the switch to fail.

Examples

None available

Responses

The following table provides explanations of the responses to the insert command.

insert (end)

Responses for the insert command

MAP output	Meaning and action
`INVALID TERMINAL IDENTIFIER`	**Meaning:** The system does ot recognize the node and terminal numbers. **Action:** Verify terminal identifier and reenter data.
`MAXIMUM LENGTH OF I/O MESSAGE ENTERED`	**Meaning:** No more message data will be accepted. **Action:** None

insertbi

Function

Use the insertbi command inserts a user-specified incoming or outgoing message into the I/O message stream with intercept and monitoring bypassed. With the insertbi command the I/O message will not be intercepted or monitored.

insertbi command parameters and variables

Command	Parameters and variables	
insertbi	in out	*node_no* [*data_string*] *terminal_no*

Parameters and variables	Description
data_string	Specifies bytes of data from 0 to 255. Up to 247 bytes of data can be entered. Data can be entered in either decimal or hexadecimal.
in	Specifies an incoming message will be inserted.
node_no	Specifies the node number for the terminal from which the message originates. Valid range is 0 through 4095.
out	Specifies that an outgoing message will be inserted.
terminal_no	Specifies the terminal number from which the message is intended. Valid range is 0 through 4095.

Qualifications

If your message is longer than the width of the terminal screen, add a plus sign (+) to the end of the line.

This command is useful for messages going to the network only.

CAUTION

INSERTing incorrect data into the I/O system can cause the switch to fail.

Examples

None available

insertbi (end)

Responses

The following table provides an explanation of the response to the insertbi command.

Responses for the insertbi command

MAP output	Meaning and action
`INVALID TERMINAL IDENTIFIER`	**Meaning:** The system does ot recognize the node and terminal numbers. **Action:** Verify terminal identifier and reenter data.

insertduplex

Function

Use the insertduplex command inserts a user-specified outgoing message to be transmitted in duplex, which means the message will be routed through both planes of the network module. PMIST supplies the message header.

insertduplex command parameters and variables

Command	Parameters and variables
insertduplex	*node_no* [*data_string*] *terminal_no*

Parameters and variables	Description
data_string	Specifies bytes of data from 0 to 255. Up to 247 bytes of data can be entered. Data can be entered in either decimal or hexadecimal.
node_no	Specifies the node number for the terminal from which the message originates. Valid range is 0 through 4095.
terminal_no	Specifies the terminal number from which the message is intended. Valid range is 0 through 4095.

Qualifications

If your message is longer than the width of the terminal screen, add a plus sign (+) to the end of the line.

CAUTION

INSERTing incorrect data into the I/O system can cause the switch to fail.

Examples

None available

Responses

The following table provides responses to the insertduplex command.

insertduplex (end)

Responses for the insertduplex command	
MAP output	**Meaning and action**
`INVALID TERMINAL IDENTIFIER`	
	Meaning: The system does ot recognize the node and terminal numbers. **Action:** Verify terminal identifier and reenter data.

insmsg

Function

Use the insmsg command to insert user-specified incoming or outgoing messages into the I/O message stream, with you having control over the contents of the I/O message header.

insmsg command parameters and variables

Command	Parameters and variables		
insmsg	*ms_card*	*ns_port*	*data_string*

Parameters and variables	Description
data_string	Specifies bytes of data entered as separate integers in the range 0 to 255. Message bytes can be entered in either decimal or hexadecimal.
ms_card	Specifies the MS card number in an ECORE/BRISC environment. This overrides the card information written to the 8-byte header which follows.
ns_port	Specifies the NS port number in an ECORE/BRISC environment. This overrides the port information written to the 8-byte header which follows.

Qualifications

CAUTION

INSERTing incorrect data into the I/O system can cause the switch to fail.

If your message is longer than the width of the terminal screen, add a plus sign (+) to the end of the line.

The significance of each byte of the message header is as follows:

Bit 7 of the I/O message tag (byte 0) specifies a message as incoming or outgoing.

Header	Significance	Remarks
byte 0	msgtag	I/O message tag:

insmsg (continued)

	bit	0	1=intercept_bypass
		1	1=duplex_msg
		2	1=route_supplied
		3	1=user_buffer
		4	1=out_of_band_reset
		5	not used
		6	not used
		7	0=incoming; 1=outgoing
byte 1	length		I/O message length: this field will be set to the actual length of the message before the message is injected; enter `0`
bytes 2,3	route		I/O message route: this field will normally be set by the I/O system; enter zeroes.
bytes 4,5,6	tid		terminal identifier byte 4: <nn: 8 lsb> byte 5: <tn: 4 msb nn: 4 msb> byte 6: <tn: 8 lsb>
byte 7	I/O error		I/O error byte; enter `0`

Note: 1 nn indicates the node number.

Note: 2 tn indicates the terminal number

Note: 3 lsb indicates least significant bits

Note: 4 msb indicates most significant bits.

Examples

None available

Responses

The following table provides explanations of the responses to the insmsg command.

insmsg (end)

Responses for the insmsg command

MAP output	Meaning and action
`INSUFFICIENT DATA FOR MESSAGE HEADER`	
	Meaning: Not all the subcommand parameters were entered.
	Action: Reenter the command and all eight parameters.
`INVALID TERMINAL IDENTIFIER`	
	Meaning: The system did not recognize the node and terminal numbers.
	Action: Verify bytes 4, 5, and 6. reenter the command and all eight parameters.

intercept

Function

Use the intercept command to enable or disable I/O message intercept or monitoring for the incoming and outgoing messages. The intercept command establishes a flow of intercepted or monitored I/O messages into the PMIST subsystem. These messages will either be stored as a Software Operating System (SOS) file, or logged in the message list if a file is not opened.

intercept command parameters and variables

Command	Parameters and variables	
intercept	in out both	[mon off on]

Parameters and variables	Description
both	Specifies interfacing or monitoring of both incoming and outgoing messages.
in	Specifies intercept or monitoring of incoming messages
mon	Enables message monitoring for specified directions. The MON setting copies the messages into the buffer without interfering with the system messaging.
off	Disables the intercept or monitoring of I/O messages and ignores messages in the specified direction.
on	Enables message intercept for the specified directions.
out	Specifies intercept or monitoring of outgoing messages

Qualifications

It is important to have the select command ON when using the intercept command.

CAUTION

Selecting INTERCEPT ON does not allow messages to reach their destination and disables the switch. Stopping I/O messages makes terminals impossible to use until a restart is done.

intercept (continued)

Examples

None available

Responses

The following table provides explanations of the responses to the intercept command.

Responses for the intercept command

MAP output	Meaning and action
`ERROR - - NO INTERCEPT ON MULTI-USER PMIST`	**Meaning:** The intercept command can only be used when on user is operating PMIST. **Action:** Send a message to other users if you need to use the intercept command.
`WARNING: SELECTED NODE/TN MSGS TO BE INTERCEPTED. SWITCH PERFORMANCE MAY DEGRADE.`	**Meaning:** The on option to the intercept command was selected. **Action:** Choose whether to continue. Enter YES or NO.
`***SELECTED NODE/TN MSGS ARE NOT BEING INTERCEPTED.`	**Meaning:** YES was entered at the prompt and interception has begun. **Action:** None.
`NO ACTION TAKEN`	**Meaning:** NO was entered at the prompt. **Action:** None.
`SELECTIVE INTERCEPT MUST BE ENABLED FOR INTERCEPT ON.`	**Meaning:** The select command was not on. **Action:** Set the select command to on and reenter the intercept command.

-continued-

intercept (end)

Responses for the intercept command (continued)

MAP output	Meaning and action
INTERCEPT IN USE	**Meaning:** The intercept facility is being used by another tool. **Action:** Send a message to other users if you need the intercept facility.

-end-

nodeno

Function

Use the nodeno command to convert the device name to the node number.

nodeno command parameters and variables

Command	Parameters and variables
nodeno	*node_type* *dev_class* *dev_no*

Parameters and variables	Description
dev_class	Specifies the class of node to be included. An example of device class is TM.
dev_no	Specifies the number of the node to be included. Valid range is 0 through 32767.
node_type	Specifies the type of node to be included. An example of node type is TM_NODE

Qualifications

None

Examples

None available

Responses

The following table provides an explanation of the response to the nodeno command.

Responses for the nodeno command

MAP output	Meaning and action
`DEVICE UNDEFINED`	**Meaning:** The system does ot recognize one or all of the parameters. **Action:** Verify data and reenter command.

pmist

Function

Use the pmist command to invoke the PMIST subsystem. Once in PMIST, the PMIST command displays the command definitions and syntax for all the PMIST commands, the current status of PMIST settings, and the list of users. If no parameters are entered, the status parameter is the default.

pmist command parameters and variables

Command	Parameters and variables
pmist	subc status users

Parameters and variables	Description
subc	Displays the explanation and command syntax for all the PMIST commands.
status	Displays the current status of the intercept, selective intercept, and I/O message recording facilities.
users	Displays all the PMIST user names and their status.

Qualifications

None

Examples

The following table provides an example of the pmist command.

pmist (continued)

Examples of the pmist command

Example	Task, response, and explanation
pmiststatus ↵ *where* status	displays the current status of intercept, selective intercept, and I/O message recording

Task: Display the status of intercept, selective intercept, and I/O message recording

Response:

```
INTERCEPT STATUS:  INCOMING MONITOR, OUTGOING MONITOR

2 NODE TYPES SPECIFIED INTERCEPT-IMMUNE

SELECTIVE INTERCEPT ENABLED; 2 TERMINAL IDENTIFIERS
SPECIFIED

NN = 0012
TN = 0136
NN = 0012
TN = 0137

I/O MESSAGE FILE OPENED, NAME = LATENITE
RECORDING ON

RESTART IS OFF

MESSAGES LOST THIS SESSION = 0
```

Explanation: The status is displayed.

Responses

The following table provides explanations of the responses to the pmist command.

Responses for the pmist command

MAP output	Meaning and action
`PMIST WAS ON DURING LAST RESTART - - NOW TURNED OFF`	**Meaning:** When the restart command is set to on, PMIST records the messages sent between a peripheral and the CC following a restart. When a user enters the PMIST subsystem following a restart, PMIST turns itself off. **Action:** Proceed with the PMIST session.

-continued-

pmist (end)

Responses for the pmist command (continued)

MAP output	Meaning and action
PMIST INITIALIZATION FAILED	**Meaning:** The MIDC utility did not initialize, and the PMIST subsystem cannot be accessed. **Action:** This message indicates a probme with PMIST or with the system. If necessary, complete a service report.
ERROR - - MAXIMUM NUMBER OF USERS IN PMIST	**Meaning:** A sixth user tried to eneter the PMISTY subsystem. **Action:** None.
FAILED TO ALLOC USER MESSAGE LIST	**Meaning:** Not enough storage is available to allocate a message list. **Action:** Proceed with observer status.
FAILED TO ALLOC CLEANUP EVENT	**Meaning:** A trap occurred or HX was entered while PMIST was operating, and events were not deallocated. **Action:** Initiate a service report.

-end-

quit

Function

Use the quit command to terminate and exit PMIST. When the quit command is issued, the intercept, select, include, and record commands are disabled, and any open SOS files are closed before PMIST terminates.

quit command parameters and variables

Command	Parameters and variables
quit	

Qualifications

None

Examples

The following table provides an example of the quit command.

Examples of the quit command

Example	Task, response, and explanation
quit ↵	
	Task: Terminate the PMIST session.
	Response: `PMIST TERMINATES`
	Explanation: The PMIST subsystem has been exited.

Responses

None

reconn

Function

Use the reconn command to reconnect a disconnected user.

reconn command parameters and variables

Command	Parameters and variables
reconn	*userid*

Parameters and variables	Description
userid	Specifies the name of a PMIST user that was previously disconnected from the present PMIST session via a disconn command.

Qualifications

The reconn command is qualified by the following exceptions, restrictions, and limitations:

- The DISCONN command must have previously been executed for this PMIST user.
- A reconn command is effective only if the specified userid applies to a user previously disconnected from the present PMIST session.

Example

The following table provides an example of the reconn command.

Examples of the reconn command

Example	Task, response, and explanation
reconn *userid* *where*	
userid	Is the name of a PMIST user that was previously disconnected from the present PMIST session via a disconn command.
	Task: Reconnect a previously disconnected user.
	Response: `RECONNECTED TO DISCONNECTED SESSION.`
	Explanation: The system reconnects this previously disconnected user to the PMIST session.

reconn (end)

Responses

The following table provides an explanation of the response to the reconn command.

Responses for the reconn command

MAP output	Meaning and action
`FAILED TO FIND DISCONN SESSION.`	**Meaning:** The DISCONN command was not previously executed for a PMIST user. **Action:** None

record

Function

Use the record command to store intercepted or monitored I/O messages in a message file.

record command parameters and variables

Command	Parameters and variables
record	open on off close

Parameters and variables	Description
close	Disables the message file. Once a message file is closed, it can never be opened and recorded to again.
off	Disables message recording to an I/O message file without closing it.
on	Enables message recording by opening an I/O message file that has not been closed.
open	Creates an I/O message file. You will be prompted to enter a device name and a new filename.

Qualifications

If I/O messages will be recorded on tape, a tape volume must be MOUNTed and formatted.

If no opened I/O message file exists, the record on command will have the same effect as the record open command.

If no parameters are entered, the current stauts of I/O message recording is displayed.

No parameters have to be used with the record command.

Recording stops if an error is encountered during file output.

Examples

None available

record (end)

Responses

The following table provides explanations of the responses to the record command.

Responses for the record command

MAP output	Meaning and action
AN OPEN I/O MESSAGE FILE EXISTS	**Meaning:** An attempt was made to open a second I/O message file for message recording. **Action:** Choose a different filename and reenter data.
NO OPEN MESSAGE FILE EXISTS	**Meaning:** An attempt was made to turn RECORD OFF or to close a nonexistent I/O message file. **Action:** Verify the name of the file to be closed, check spelling, and reenter the data.
INVALID FILE IDENTIFIER	**Meaning:** Device name was not entered, or a null line was entered. **Action:** Verify the device name, such as SFDEV, and the new file name, and reenter data.
INVALID DEVICE PARAMETER	**Meaning:** Device name entered is not valid. **Action:** Verify the device name, check spelling, and reenter data.
INVALID FILE NAME, RE-ENTER COMMAND	**Meaning:** Self-explanatory **Action:** Verify device and filename and reenter data.

remove

Function

Use the remove command to delete designated terminal identifiers from the selective intercept list.

remove command parameters and variables

Command	Parameters and variables		
remove	nid	*node_no*	*terminal_no*
	mta	*node_no*	*terminal_no*
	all		

Parameters and variables	Description
all	Deletes the entire list of TIDs being monitored.
mta	Deletes the Message Transport Address option, which adds 4096 to the node number being entered with the remove command. This allows messages that use the Message Transport System protocol to be removed.
nid	Deletes the Network Identifier option, which adds 4096 to the node number being entered with the remove command. This allows messages that use the Message Transport System protocol to be removed.
node_no	Specifies the node number. Valid range is 0 through 4095.
terminal_no	Specifies the terminal number. Valid range is 0 through 4095.

Qualifications

More than one node number and terminal number pair can be entered with the remove command. For example:

```
REMOVE 15 36 15 39
```

If the remove command is entered with no parameters, the result of the last convert command will be removed from the list of TIDs to be monitored.

Examples

None available

Responses

The following table provides explanations of the responses to the remove command.

remove (end)

Responses for the remove command

MAP output	Meaning and action
`INVALID TERMINAL IDENTIFIER`	**Meaning:** The system did not recognize the terminal identifier. **Action:** Verify node and terminal number and reenter data.
`CLEARING ALL TIDS`	**Meaning:** The remove all command has been processed and the entire list of TIDs to monitor is erased. **Action:** None

restart

Function

Use the restart command to resume message recording during a system restart.

restart command parameters and variables

Command	Parameters and variables		
restart	off		
	on	*device*	*file_name*

Parameters and variables	Description
device	Specifies the device to be included. An example of a device is SFDEV.
file_name	Specifies the file where the messages will be sent after the restart.
off	Deactives restart.
on	Deactivates RESTART. After you enter the restart on command, a prompt to enter the device and filenames will appear.

Qualifications

If no parameter is specified, the current restart status is displayed.

A restart causes any PMIST monitoring and recording to be stopped. However, if a restart has been activated, then PMIST recording is initiated during the restart using the intercept and include status that was in effect when the restart on command was issued. Messages are recorded in the SFDEV file specified in the restart command.

Examples

None available

Responses

The following table provides explanations of the responses to the restart command.

restart (end)

Responses for the restart command

MAP output	Meaning and action
`ERROR - - NO RESTART IN MULTI-USER PMIST`	**Meaning:** The RESTART command can be used only when there is one user operating PMIST. **Action:** Message other users if you need to activate the restart command.
`ERROR - - NOT ENOUGH MEMORY FOR SELECTED LIST`	**Meaning:** Memory is not large enough for the list of terminal identifiers returned. **Action:** Delete as many terminal identifiers as necessary.
`ERROR - - NOT ENOUGH MEMORY FOR RESTART INFO - - REMOVE SOME TIDS`	**Meaning:** Memory is not large enough for the list of terminal identifiers returned. **Action:** Delete as many terminal identifiers as necessary.

select

Function

Use the select command to allow only the I/O messages specified by you to be intercepted. I/O messages to be intercepted are specified by you with the include and remove commands.

select command parameters and variables

Command	Parameters and variables
select	on off

Parameters and variables	Description
off	Disables selective intercept.
on	Enables selective intercept.

Qualifications

If no parameters are entered, the current status of selective intercept and the user-specified list of selected terminal identifiers is displayed.

The select command is on by default.

Examples

None available

Responses

The following table provides explanations of the responses to the select command.

Responses for the select command

MAP output	Meaning and action
`ERROR - - SELECT MAY NOT BE TURNED OFF IN MULTI-USER PMIST`	**Meaning:** The select command can be disabled onlky if one user is operating in the PMIST subsystem. **Action:** Send a message to the other users if you need select off.

-continued-

select (end)

Responses for the select command (continued)

MAP output	Meaning and action
`INTERCEPT MUST BE OFF OR MON BEFORE SELECT OFF ALLOWED`	**Meaning:** The select command cannot be disabled unles the intercept command is disabled. **Action:** Set intercept to mon or off.
-end-	

terminate

Function

Use the terminate command to stop the IMD process, terminate and exit PMIST. The terminate command allows PMIST, IMD, MIDC, and MDIOMSG modules to be unloaded, respectively.

terminate command parameters and variables

Command	Parameters and variables
terminate	There are no parameters or variables.

Qualifications

None

Examples

None available

Responses

The following table provides explanations of the responses to the terminate command.

Responses for the terminate command

MAP output	Meaning and action
`THIS WILL DISABLE INTERCEPTING`	**Meaning:** Processing the TERMINATE command stops all message interception. **Action:** Choose whether to continue (YES/NO prompt)
`THERE ARE OTHER USERS . . . PROCEEDING WILL STOP THEM`	**Meaning:** There is more than one user usng PMIST. Processing the terminate command ends their PMIST sessions as well. **Action:** Choose whether to terminate the session. Enter YES or NO at the prompt.
`NO ACTION TAKEN`	**Meaning:** NO was entered at the prompt. PMIST is not terminated. **Action:** None.

-continued-

terminate (end)

Responses for the terminate command (continued)

MAP output	Meaning and action
PMIST TERMINATES	**Meaning:** PMIST is exited. **Action:** None

-end-

tmsglist

Function

Use the tmsglist command to advance the pointer cursor to the top of the intercepted messages list.

tmsglist command parameters and variables

Command	Parameters and variables
tmsglist	There are no parameters or variables.

Qualifications

None

Examples

None available

Responses

None available

verify

Function

Use the verify command to verify a message sequence stored in a file. Injected and intercepted messages are compared with the filed messages until a timeout occurs, an error condition occurs, the complete message sequence has been successfully verified, or an intercepted message does not match the expected message.

verify command parameters and variables

Command	Parameters and variables	
verify	*file_name*	in out

Parameters and variables	Description
file_name	Specifies the name of the file to be verified.
in	Specifies incoming messages will be verified.
out	Specifies outgoing messages will be verified.

Qualifications

When the verify command is invoked, the intercepted messages are altered as the message sequence is verified. Intercepts are not returned to their original state when the verify is completed, and they are disabled. This may affect message recording if the record command is enabled.

Message recording and verifying require two storage devices if they will be used concurrently, with one being SFDEV.

Examples

None available

Responses

The following table provides explanations of the responses to the verify command.

verify (continued)

Responses for the verify command

MAP output	Meaning and action
`ERROR - - VERIFY IS NOT AVAILABLE IN MULTI-USER PMIST`	**Meaning:** This command can be used only when one user is using PMIST. **Action:** Send a message to the other users if you need the verify command.
`FILE SPECIFIED NOT A MESSAGE FILE`	**Meaning:** The filename entered with the verify command is not an I/O message file. **Action:** Verify name of file and spelling and reenter the subcommand and filename.
`INVALID FILENAME SPECIFIED`	**Meaning:** Self-explanatory **Action:** Verify filename, check spelling, and reenter command.
`MAILBOX COULD NOT BE ALLOCATED`	**Meaning:** Internal processes cannot be processed because a mailbox could not be allocated. **Action:** Wait for a while and try the command again. If this message appears again, exit PMIST and try again later. If this message reappears each time you use PMIST, initiate a service report.
`INTERCEPT IN USE`	**Meaning:** The intercept facility is being used by another tool. **Action:** Send a message to the other users if you need the INTERCEPT facility.
`MESSAGE SEQUENCE SUCCESSFULLY VERIFIED`	**Meaning:** Specified record was successfully read in from the message file. **Action:** None

-continued-

verify (end)

Responses for the verify command (continued)

MAP output	Meaning and action
`INVALID TERMINAL IDENTIFIER`	**Meaning:** The system did not recognize the terminal identifier. **Action:** Verify terminal identifier and reenter the command.

`NO INTERCEPTED I/O MESSAGE RECEIVED:-`
`EITHER WAITFAILED,`
`OR WAITTIMEOUT {90,SECS},`
`OR NO BUFFERS AVAILABLE FOR LONG`
`INTERCEPTED I/O MESSAGES.`
`EXPECTED I/O MESSAGE:`
	Meaning: The sequence that was verified was not the same as the recorded sequence.
	
	Action: Display the message sequence that was verified to find where the messages do not match.

-end-	

XPMIST level commands

Use the XPMIST level of the MAP to debug integrity problems caused by call processing.

The XPMIST subsystem is a version of PMIST.

XPMIST contains many of the same commands as PMIST, except XPMIST has several commands that are not contained in the PMIST subsystem.

Note: The PMIST commands intercept on, and select off, are disallowed in XPMIST.

The XPMIST utility commands are made available to you following successful access to the XPMIST subsystem.

Accessing the XPMIST level

To access the XPMIST level, enter the following from the CI level:

xpmist ↵

XPMIST commands

The commands available at the XPMIST MAP level are described in this chapter and arranged in alphabetical order. The page number for each command is listed in the following table.

XPMIST commands

Command		Page
associate	XPMIST	5-3
extract	XPMIST	5-5
includemtce	XPMIST	5-9
includenode	XPMIST	5-11
integ	XPMIST	5-13

-continued-

XPMIST commands (continued)

Command		Page
mcount	XPMIST	5-15
remove	XPMIST	5-17
removemtce	XPMIST	5-19
removenode	XPMIST	5-21
xpmist	XPMIST	5-23

-end-

associate

Function

Use the associate command to allow all the messages associated with a call to be monitored. This command monitors messages to or from terminals that have not been designated for associative intercept/monitoring, but are associated with a call involving terminals that are designated for associative intercept.

associate command parameters and variables

Command	Parameters and variables
associate	off on dis ena

Parameters and variables	Description
dis	Disables the associate command.
ena	Enables the associate command. The command must be enabled before it can be turned on.
off	Turns the associate command off.
on	Turns the associate command on if the command has been enabled.

Qualifications

None

Examples

None available

Responses

The following table provides explanations of the responses to the associate command.

associate (end)

Responses for the associate command

MAP output	Meaning and action
ASSOCIATIVE INTERCEPT MUST BE ENABLED.	**Meaning:** The associate on command cannot be entered until the associate command is enabled. **Action:** Enter associate ena. Reenter associate on.
STORE UNAVAILABLE FOR ASSOCIATIVE INTERCEPT DATA	**Meaning:** Not enough storage is available to monitor all the messages associated with a call. **Action:** Use the includenode command. Exit XPMIST and attempt to use the associate command at another time.

extract

Function

Use the extract command to display the I/O messages that occurred between the user-specified start and stop times. You can use this command to look at a specific sequence of messages within a large message list.

extract command parameters and variables

Command	Parameters and variables		
extract	open	*filename*	
	close		
	callid	*call_id*	
	from	*hours*	*minutes*
	to	*hours*	*minutes*
	for	*no_of_msgs*	
	integ		
	mcount	*node_no*	
	tid	*node_no*	*terminal_no*

Parameters and variables	Description
callid	Extracts all messages with the specified call identifier. A CALLID uniquely identifies a call, identifies resources associated with a call, and aids in identifying and recovering a call.
callid	Specifies the call identifier (CALLID). The CALLID is composed of an index into the Call Condensed Block (CCB) and a sequence number from the Terminal Linkage Block.
close	Closes a message file that has been opened
filename	Specifies the name of the message file you want to open.
for	Specifies the number of messages you want displayed. For example, if you want to look at only six messages, you would enter EXTRACT FOR 6. If no number is entered, the default is 000. The number of messages is from 0 to 32767.
from	Indicates that all messages recorded before the time specified with this parameter will not be displayed. The default is 00:00:00.0. For example, if you performed a XPMIST trace that began at 18:54:31.5 and ended at 19:32:32.6, you can view the messages that occurred FROM 19:00:00 onward.
hours	Specifies the time in hours that the extracted message begins or ends. The valid range is 0 through 99.

-continued-

extract (continued)

extract command parameters and variables (continued)

Parameters and variables	Description
integ	Extracts all Integrity messages as well as all messages for calls related to the Integrity messages. The record file is scanned three times as follows: ▪ Scan Pass 1: Scan for Integrity Lost or Integrity Fail messages. Store the CALLIDs with the Integrity messages. Save the node and channel data for Integrity Fail messages associated with nil CALLIDs. ▪ Scan Pass 2: If any Integrity Fail messages have nil CALLIDs, scan all outgoing messages for ASSCH or ASSCH2 primitives to determine which channels these messages were associated with. Save the CALLIDs that were associated with these channels prior to the integrity failure. ▪ Display any message from the lists compiled in steps 1 and 2 that contains a valid CALLID. Use the INTEG parameter in conjunction with the TO and FROM parameters.
mcount	Extracts the record of counted messages from the message file. The MCOUNT subcommand counts how many messages were recorded and stores the count in the user's message file. The MCOUNT option of the EXTRACT command extracts the message count from a message file. Use the MCOUNT parameter in conjunction with the TO and FROM parameters.
minutes	Specifies the time in minutes that the extraced message begins or eneds. Valid range is 0 through 59.
no_of_msgs	Specifies the number of messages to be displayed. Valid range is 0 through 32767.
node_no	Identifies the node whose messages are extracted. The valid range is from 0 through 4095.
open	Opens a message file that already exists
seconds	Specifies the time in seconds that the extracted message begins or ends. The valid range is 0 through 59.
tenths_sec	Specifies the time in tenths of seconds that the extracted message begins or ends. The valid range is 0 through 9.
terminal_no	Identifies the terminal whose messages are extracted. The valid range is 0 through 4095.

-continued-

extract (continued)

extract command parameters and variables

Parameters and variables	Description
tid	Specifies the terminal identifier. This parameter allows you to view those messages that were sent to and from one particular terminal. Only messages to and from this terminal will be extracted. The TID numbers are from 0 to 4095.
to	Indicates that all messages recorded after the time specified with this parameter will not be displayed. For example, if you performed a XPMIST trace that began at 18:54:31.5 and ended at 19:32:32.6, you can view the messages that occurred FROM 18:54:31.5 TO 19:10:54.3. If no time is entered, the default is 99:59:59.9.

-end-

Qualifications

None

Examples

None available

Responses

The following table provides explanations of the responses to the extract command.

Responses for the extract command

MAP output	Meaning and action
`FILE SPECIFIED NOT A MESSAGE FILE`	**Meaning:** Self-explanatory **Action:** Verify message filename, check filename spelling, and reenter filename.
`NO EXTRACT FILE OPENED`	**Meaning:** A filename or the OPEN parameter was not entered. **Action:** Reenter open parameter with a filename.

-continued-

extract (end)

Responses for the extract command (continued)

MAP output	Meaning and action
`END OF FILE ENCOUNTERED`	**Meaning:** No time was specified with the to parameter. **Action:** None
`INCOMPATIBLE OPTIONS TID, MCOUNT, INTEG`	**Meaning:** The parameters tid, callid, integ, and mcount cannot be used simultaneously. **Action:** Choose one of the options and reenter data.
`INTEG SCAN PASS1`	**Meaning:** Scaning for integrity lost or integrity lost or integrity fail messages is under way. **Action:** None
`INTEG SCAN PASS2`	**Meaning:** If any integrity fail messages have nil callids, scanning for all outgoing mesage for ASSCH or ASSCH2 primitives is under way. **Action:** None
`DISPLAY EXTRACTED MSGS`	**Meaning:** Messages containing a valid callid are displayed. **Action:** None

-end-

includemtce

Function

Use the includemtce command to allow the maintenance terminal (terminal 0) of a node to be monitored. This command can be used to select a unit (unit 0 or unit 1) to be monitored. For example, messages for the active unit can be recorded while the inactive unit is loading.

includemtce command parameters and variables

Command	Parameters and variables
includemtce	*node_no* *ext_byte*

Parameters and variables	Description
ext_byte	Specifies the extension byte identifying the unit number within the node. Valid entries are 0 or 1.
node_no	Specifies the node number. Valid range is 0 to 4095.

Qualifications

None

Examples

None available

Responses

The following table provides explanations of the responses to the includemtce command.

Responses for the includemtce command

MAP output	Meaning and action
`INVALID TERMINAL IDENTIFIER`	**Meaning:** The system does not recognize the node number entered. **Action:** Verify node number and reenter data.
`SELECTED TIDS LIST FULL`	**Meaning:** Memory is not large enough for the list of terminal identifiers selected. **Action:** Delete as many terminal identifiers as necessary.

includenode

Function

Use the includenode command to allow the maintenance terminal (terminal 0) of a node to be monitored. This command can be used to select a unit (unit 0 or unit 1) to be monitored. For example, messages for the active unit can be recorded while the inactive unit is loading.

includenode command parameters and variables

Command	Parameters and variables
includenode	*node_no*

Parameters and variables	Description
node_no	Specifies the node number. Valid range is 0 to 4095.

Qualifications

None

Examples

None available

Responses

The following table provides explanations of the responses to the includenode command.

Responses for the includenode command

MAP output	Meaning and action
`INVALID TERMINAL IDENTIFIER`	**Meaning:** The system does not recognize the node number entered. **Action:** Verify node number and reenter data.
`SELECTED TIDS LIST FULL`	**Meaning:** Memory is not large enough for the list of terminal identifiers selected. **Action:** Delete as many terminal identifiers as necessary.

integ

Function

Use the integ command to monitor Integrity Fail messages associated with a monitored call.

integ command parameters and variables

Command	Parameters and variables
integ	off on

Parameters and variables	Description
off	Disables the integ command.
on	Enables the integ command.

Qualifications

All Integrity Fail messages are monitored unless they are not related to the call being monitored.

If the associate command is off, all Integrity Fail messages are monitored.

If the associate command is on and an Integrity Fail message is received, the following events occur:

The channel information is extracted.

The channel is mapped to the associated Virtual Identifier (VID) or Terminal Identifier (TID) .

> ***Note:*** A Terminal Identifier (TID) defines a node and a terminal. Devices attached to a particular terminal (such as a business set) are defined by Virtual Identifiers (VID).

If the terminal state for the VID is LINKEDTOCPTLB or MULTICPLINKED, the call identifier (CALLID) is saved.

> ***Note:*** Terminal state LINKEDTOCPTLB indicates a call is linked to the terminal.

> ***Note:*** Terminal state MULTICPLINKED indicates the terminal is linked to two calls. Only one call is active at a time.

integ (end)

> ***Note:*** The callid uniquely identifies a call. The CALLID is composed of an index to the Call Condense Block (CCB) and a sequence number into the Terminal Linkage Block.

If the callid is not associated with any of the selected Terminal Identifiers (TID), the Integrity Fail message is discarded.

The following counts are viewed by entering integ with no parameters:

- all Integrity Lost messages
- all Integrity Fail messages
- all Integrity Fail messages with valid callid

Examples

None available

Responses

None available

mcount

Function

Use the mcount command to count messages and make a record of the count for each selected node.

mcount command parameters and variables

Command	Parameters and variables
mcount	on off saveon saveoff period *unit*

Parameters and variables	Description
off	Disables message counting
on	Enables message counting
period	Specifies the time during which messages are counted
saveoff	Discards the messages being counted
saveon	Saves the messages being cunted along with the message count
unit	Indicates the time unit. Valid entries are: sec for a second min for a minute hour for an hour

Qualifications

If no parameter is entered with the mcount command, the current status of counting is displayed. The following is an example of the mcount status display:

```
MCOUNT: ON   SAVE:  OFF    PERIOD: MIN
```

Examples

None available

Responses

None available

remove

Function

Use the remove command to remove the terminal identifier from selective intercept or monitoring.

remove command parameters and variables

Command	Parameters and variables
remove	*node_no* *terminal_no*

Parameters and variables	Description
node_no	Specifies the node number. Valid range is 0 to 4095.
terminal_no	Specifies the terminal number. Valid range is 0 to 4095.

Qualifications

More than one node number and terminal number pair can be entered with the REMOVE subcommand. For example:

```
REMOVE 15 36 15 39
```

Examples

None available

Responses

The following table provides an explanation of the response to the remove command.

Responses for the remove command

MAP output	Meaning and action
`INVALID TERMINAL IDENTIFIER`	**Meaning:** The system does not recognize the node number entered. **Action:** Verify node number and reenter data.

removemtce

Function

Use the removemtce command to remove the maintenance terminal (terminal 0) from monitoring or interception. This command removes the unit (unit 0 or unit 1) from monitoring or interception.

removemtce command parameters and variables

Command	Parameters and variables
removemtce	*node_no* *ext_byte*

Parameters and variables	Description
ext_byte	Specifies the extension byte identifying the unit number within the node. Valid entries are 0 or 1.
node_no	Specifies the node number.

Qualifications

None

Examples

None available

Responses

The following table provides an explanation of the response to the removemtce command.

Responses for the removemtce command

MAP output	Meaning and action
`INVALID TERMINAL IDENTIFIER`	**Meaning:** The system does not recognize the node number entered. **Action:** Verify node number and reenter data.

removenode

Function

Use the removenode command to remove all terminals on a node from monitoring or interception. This command is not related to the PMIST remove command.

removenode command parameters and variables

Command	Parameters and variables
removenode	*node_no*

Parameters and variables	Description
node_no	Specifies the node number. Valid range is 0 to 4095.

Qualifications

None

Examples

None available

Responses

The following table provides explanations of the responses to the removenode command.

Responses for the removenode command

MAP output	Meaning and action
`INVALID TERMINAL IDENTIFIER`	**Meaning:** The system does not recognize the node number entered. **Action:** Verify node number and reenter data.

xpmist

Function

Use the xpmist command to invoke the XPMIST subsystem. Once in XPMIST the xpmist subc command displays the PMIST commands or the current status.

xpmist command parameters and variables

Command	Parameters and variables
xpmist	subc status

Parameters and variables	Description
status	Displays the current status of the intercept, selective intercept, and I/O message recording facilities.
subc	Displays the PMIST commands or the current status of the PMIST commands.

Qualifications

None

Examples

None available

Responses

The following table provides an explanation of the response to the xpmist command.

Responses for the xpmist command

MAP output	Meaning and action
`FAILED TO ALLOC USER MESSAGE LIST`	**Meaning:** Not enough storage is available to allocate a message list. **Action:** Proceed with observer status.

PUPI level commands

Use the PUPI level of the MAP to access the PMIST subsystem and set up normal message tracking. PUPI defines shortened forms of two PMIST commands (dmsglist and cmsglist) and contains commands for tracing execs, for dumping variable areas, and for calculating node and terminal numbers.

Accessing the PUPI level

To access the PUPI level, enter the following from the CI level:

read pupi ↵

If your switch contains the PUPI file, you can enter PMIST by typing the above. When this command is processed, the pmist multi user prompt will appear 35 times in response to the CI commands PUPI is processing. You can continue once the prompts stop.

Figure 6-1 shows the PUPI file listing.

If the switch does not recognize the read pupi command, type

listsf all ↵

to list the files stored in SFDEV. If the PUPI file is not listed, it has not been loaded into the switch.

Note: The PUPI file may have a slightly different name in each switch, such as PUPIFILE instead of PUPI. When you list the files stored in SFDEV , look for variations in the name of the file.

The PUPI commands are available following successful access to the exec when read pupi is entered and PMIST is invoked. To produce a listing of all PUPI commands, type

print pupi ↵

Figure 6-1xxx
PUPI exec (Part 1 of 2)

```
PMIST
INTERCEPT BOTH MON
COMMAND CL CMSGLIST
COMMAND D DMSGLIST
COMMAND SLP (SLEEP 1 SECS; REMOVE @1 0)
COMMAND TR (INSERT OUT @1 0 #81 @3 @2 #62)
COMMAND STOP (INCLUDE @1 0;INSERT OUT @1 0 #81 #A 0 5 +
    #80 @2  #63 8; SLP @1)
COMMAND VAR (INCLUDE @1 0;INSERT OUT @1 @2 #81 #A #1 6 #9F 0 #BF+
    #9F #10 #BF #93 #20 #B3 8;SLP @1 )
COMMAND REF (INCLUDE @1 0;INSERT OUT @1 @2 #81 #A #2 6 #81 #E 0 +
    #55 #BD 8; SLP @1 )
COMMAND SCPTS (INCLUDE @1 0; INSERT OUT @1 @2 #81 #A #3 6 +
    #50 #51 #B1 8; SLP @1)
COMMAND PMDD (INCLUDE @1 0;INSERT OUT @1 0 #84 @2 @3 @4 #A #4 5 +
    #54 8; SLP @1)
COMMAND PMDUMP (INCLUDE @1 0;STOP @1 @3;VAR @1 @2;REF @1 @2; +
    SCPTS @1 @2;SLEEP 1 SECS ;REMOVE @1 0);
COMMAND BLK (PRINT ' ')
COMMAND THEAD (PRINT 'TRMNL=')
COMMAND LMNT (NODENO LM_NODE LM (@1*2+(@2)); THEAD;  +
    PRINT (1+(@3*32+(@4)));BLK)
COMMAND R120 (Z9/120->TEMP;TEMP+Z9-(TEMP*120)->Z9)
COMMAND SPCASE (IF (Z8 = 3690) THEN (PRINT 120) ELSE (PRINT Z9))
COMMAND DCMT (THEAD;((-30)+(31*(@1*24+(@2))))->Z9;Z9->Z8;  +
    R120;R120;SPCASE;BLK)  %% SPCASE IS DCMT 4 24 = 120
COMMAND PORTCHNL (@1 * 32 + (@2) + 1)
COMMAND IDTCPORTCHNL ((PORTCHNL @1 @2) -> Z7; +
                  IF (@2 > 15) THEN ( PRINT (Z7 + 1)) +
                                ELSE (PRINT Z7))
COMMAND DCMNT (NODENO DCM_NODE DCM @1;DCMT @2 @3)
COMMAND TM8NT (NODENO TM_NODE TM8 @1;THEAD;PRINT (1+(@2));BLK)
COMMAND MTMNT (NODENO TM_NODE MTM @1;THEAD;PRINT (1+(@2));BLK)
COMMAND DTCNT (NODENO LTC_NODE DTC @1;THEAD; +
        PRINT (PORTCHNL @2 @3);BLK)
COMMAND IDTCNT (NODENO LTC_NODE IDTC @1;THEAD; +
    PRINT (IDTCPORTCHNL @2 @3);BLK)
COMMAND PDTCNT (NODENO LTC_NODE PDTC @1;THEAD; +
        PRINT (PORTCHNL @2 @3);BLK)
COMMAND COM1 ( +
PRINT 'START EXEC TRACE: TR     <NODE> <TERM> <BUFFER 0/1>'; +
PRINT 'STOP  EXEC TRACE: STOP   <NODE> <BUFFER 0/1>'; +
PRINT 'DUMP PP VAR AREA: VAR    <NODE> <TERM>'; +
```

Figure 6-1xxx
PUPI exec (Part 2 of 2)

```
PRINT 'DUMP REFLEX AREA: REF    <NODE> <TERM>')
COMMAND COM2 ( +
PRINT 'GET  SCAN PTS:    SCPTS  <NODE> <TERM>'; +
PRINT 'ALL  INFO ABOVE:  PMDUMP <NODE> <TERM> <BUFFER 0/1>'; +
PRINT 'DUMP PM STORE:    PMDD   <NODE> <ADDR LSB> <ADDR MSB>
                                                 <NBYTE>';+
BLK)
COMMAND COM3 ( +
PRINT 'LM  NODE/TERM:   LMNT   <FRAME>  <BAY>  <DRAWER> <CKT>'; +
PRINT 'DCM NODE/TERM:   DCMNT  <DCM_NO> <CARD> <SLOT>';     +
PRINT 'TM8 NODE/TERM:   TM8NT  <TM8_NO> <CKT>';             +
PRINT 'MTM NODE/TERM:   MTMNT  <MTM_NO> <CKT>';             +
PRINT 'DTC NODE/TERM:   DTCNT  <DTC_NO> <CARRIER> <CKTNO>'; +
PRINT 'IDTC NODE/TERM:  IDTCNT <IDTC_NO> <CARRIER> <CKTNO>'; +
PRINT 'PDTC NODE/TERM:  PDTCNT <PDTC_NO> <CARRIER> <CKTNO>'; +
BLK)
COMMAND QPUP (COM1;COM2;COM3)
BLK
PRINT 'TYPE QPUP FOR COMMAND SYNTAX; EDIT PUPI FOR NOTES'
READ PREVIOUS
                    PUPI NOTES
  (1) TO GET THE NODENO OF AN LM/RLM IN AN OFFICES WITH RLM:
         ENTER THE PM LEVEL OF THE MAP
         SELECT THE LM/RLM
         DO A QUERY OR QUERYLM
  (2) FOR TRUNKS ON AN RSM USE THE NODENO OF THE ASSOCIATED RLM.
      ADD 610 TO THE CKT TO GET THE TRMNL NO.  TERMINAL 610 IS THE
      ADDRESS OF THE RSM CPU.
  (3) USE TRACE BUFFER 0 WITH TR/STOP FOR LM/RLM/RSM.
```

Abbreviated PMIST commands

If you enter PMIST with the read pupi command, you can use the abbreviation of CMSGLIST, as follows:

CL

If you enter PMIST with the read pupi command, you can also use the abbreviation of DMSGLIST, as follows:

D

PUPI commands

The commands available at the PUPI MAP level are described in this chapter and arranged in alphabetical order. The page number for each command is listed in the following table.

PUPI commands

Command		Page
dcmnt	PUPI	6-5
dtcnt	PUPI	6-7
idtcnt	PUPI	6-9
lmnt	PUPI	6-11
mtmnt	PUPI	6-13
pdtcnt	PUPI	6-15
pmdd	PUPI	6-17
pmdump	PUPI	6-21
qpup	PUPI	6-25
ref	PUPI	6-27
scpts	PUPI	6-29
stop	PUPI	6-33
tm8nt	PUPI	6-37
tr	PUPI	6-39
var	PUPI	6-43

-end-

dcmnt

Function

Use the dcmnt command to calculate the node and terminal number for a Digital Carrier Module (DCM).

dcmnt command parameters and variables

Command	Parameters and variables		
dcmnt	*dcm_no*	*card*	*slot*

Parameters and variables	Description
card	Specifies the DCM card position
dcm_no	Specifies the DCM card position
slot	Specifies the DCM timeslot

Qualifications

None

Examples

None available

Responses

None available

dtcnt

Function

Use the dtcnt command to calculate the node and terminal numbers for Digital Trunk Controllers (DTC).

dtcnt command parameters and variables

Command	Parameters and variables		
dtcnt	*dtc_no*	*carrier*	*cktno*

Parameters and variables	Description
carrier	Specifies the DTC card position
cktno	Specifies the DTC circuit number
dtc_no	Specifies the identification numer for the DTC

Qualifications

None

Examples

None available

Responses

None available

idtcnt

Function

Use the idtcnt command to calculate the node and terminal numbers for International Digital Trunk Controller (IDTC).

idtcnt command parameters and variables

Command	Parameters and variables		
idtcnt	*idtc_no*	*carrier*	*cktno*

Parameters and variables	Description
carrier	Specifies the IDTC card position
cktno	Specifies the IDTC circuit number
idtc_no	Specifies the identification number for the IDTC

Qualifications

None

Examples

None available

Responses

None available

lmnt

Function

Use the lmnt command to calculate the node and terminal numbers for Line Modules (LM) and Remote Line Modules (RLM).

lmnt command parameters and variables

Command	Parameters and variables			
lmnt	*frame*	*bay*	*drawer*	*ckt*

Parameters and variables	Description
bay	Specifies the bay number
ckt	Specifies the LMNT circuit number
drawer	Specifies the drawer number
frame	Specifies the frame number

Qualifications

None

Examples

None available

Responses

None available

mtmnt

Function

Use the mtmnt command to calculate the node and terminal numbers for Maintenance Trunk Modules (MTM).

mtmnt command parameters and variables

Command	Parameters and variables	
mtmnt	*mtm_no*	*ckt*

Parameters and variables	Description
ckt	Specifies the MTM circuit position
mtm_no	Specifies the identification number for the MTM

Qualifications

None

Examples

None available

Responses

None available

pdtcnt

Function

Use the pdtcnt command to calculate the node and terminal numbers for PCM30 Digital Trunk Controller (PDTC).

pdtcnt command parameters and variables

Command	Parameters and variables		
pdtcnt	*pdtc_no*	*carrier*	*cktno*

Parameters and variables	Description
carrier	Specifies the PDTC card position
cktno	Specifies the PDTC circuit number
pdct_no	Specifies the PDTC identification number

Qualifications

None

Examples

None available

Responses

None available

pmdd

Function

Use the pmdd command to insert data for the specified area into a report and sends it to the CC. This command can be used to dump the first 27 bytes of the terminal process block not accessible from the stack primitives.

pmdd command parameters and variables

Command	Parameters and variables
pmdd	*node_no* *addr_lsb* *addr_msb* *nbytes*

Parameters and variables	Description
addr_lsb	Specifies the least significant byte of the start address
addr_msb	Specifies the most significant byte of the start address
nbytes	Specifies the area of data to be dumped. 40 bytes is recommended.
node_no	Specifies the PM node number.

Qualifications

Calculate the start address for TM and DCM as follows:

START ADDRESS = BASE ADDRESS + ((TERMINAL # - 1) * #40)

BASE ADDRESS = #4700 for TM
#8100 for DCM

For instance, if the terminal number of a TM8 is equal to 1, the start address would be calculated as follows:

START ADDRESS = #4700 + ((1-1)* #40)

START ADDRESS = #4700 + (0 * #40)

START ADDRESS = #4700

In this example, the least significant byte of the start address is #00, and the most significant byte is #47.

Sample Session with the PMDD command

Following is an example of the output produced by the pmdd command.

pmdd (continued)

The include command allows PMIST to display the outgoing message produced by the pmdd command. The include command is not a requirement for the pmdd command.

```
PMIST MULTI USER:
INTERCEPT BOTH MON
PMIST MULTI USER:
TM8NT 3 0
NODENO=31
TRMNL=
1

PMIST MULTI USER:
INCLUDE 31 1
PMIST MULTI USER:
CMSGLIST
PMIST MULTI USER:
PMDD 31 0 #47 40
PMIST MULTI USER:
DMSGLIST

OUTGOING  15:16:12.1  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=11
84   00  47  20  0A  04  05  54  08

00          84        STKI5          00 47 20 0A 04
06          05        OPRPT
07          54        INCM
08          08        CLRPT

INCOMING  15:16:12.2  NODE TYPE= TM_NODE   TEST_ACK_MSG
NN= 001F  TN= 0000  MSGTAG= 3B  ROUTE= 0080  ERROR= 00  LENGTH=2D
0A 04  20  00  47  00  00  00  03  7F  63  A0  00  00  00  00  00  00  00  01  80  00
00  10  00  00  00  00  00  2F  00  00  A6  10  35  35  54  00  0C  00  00  5B  00  00
00
END OF INTERCEPTED I/O MESSAGES.
PMIST MULTI USER:
```

In the previous example, the outgoing message contains the instructions sent to terminal 0 of the TM8 by the pmdd command.

The STKI5 primitive stacks the five bytes required by the OPRPT primitive, which are the start address least significant byte (00), the start address most significant byte (47), the number of bytes to be dumped (20), and the message type (0A 04).

The OPRPT primitive opens a report. The INCM primitive includes the specified memory in the open report.

The CLRPT primitive closes the report and queues it for output.

The incoming message is sent from terminal 0 and contains the message type (0A 04) and the 40 bytes of memory.

Examples

See above example.

Responses

See above example.

pmdump

Function

Use the pmdump command to perform all of the functions of the stop, var, ref, and scpts commands.

pmdump command parameters and variables

Command	Parameters and variables		
pmdump	*node*	*term*	*buffer*

Parameters and variables	Description
buffer	Specifies the trace buffer number. Valid entries are 0 or 1.
node	Specifies the node number.
term	Specifies the terminal number.

Qualifications

Sample session with the PMDUMP command

Following is an example of the output produced by the pmdump command. In this example, each outgoing message is identified as either A, B, C, or D. Output message A is produced by the stop command. Output message B is produced by the var command. Output message C is produced by the ref command. Output message D is produced by the scpts command.

Each outgoing message in this example is labelled to clarify the pmdump output. The pmdump command does not produce labels next to the outgoing messages.

The include command allows PMIST to display the outgoing message produced by the pmdump command. The include command is not a requirement for the pmdump command.

```
PMIST MULTI USER:
INTERCEPT BOTH MON
PMIST MULTI USER:
TM8NT 3 0
NODENO=31
TRMNL=
1
```

pmdump (continued)

```
PMIST MULTI USER:
INCLUDE 31 1
PMIST MULTI USER:
CMSGLIST
PMIST MULTI USER:
PMDUMP 31 1 0
PMIST MULTI USER:
DMSGLIST

A) OUTGOING  15:17:54.3  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=10
81 0A 00 05 80 00 63 08

00          81        STKI2          0A 00
03          05        OPRPT
04          80        STKI1          00
06          63        TRARP
07          08        CLRPT

INCOMING  15:17:54.3  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 3B  ROUTE= 0080  ERROR= 00  LENGTH=2C
0A 00  01  17  44  06  4B  37  8D  3A  A4  32  A5 4E  A3  32  44  06  4B  37  8D 3A
A4 32  A5 4E A3  32  FF  FF  FF  FF  FF  FF  FF FF

B) OUTGOING  15:17:55.6  NODE TYPE= TM_NODE
NN= 001F  TN= 0001  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=16
AGENT= CKT                OTMF2    1
81  0A  01  06  9F  00  BF  9F  10  BF  93  20  B3  08

00          81        STKI2          0A 01
03          06        OPMRPT
04          9F        STKVG          00
06          BF        INCG
07          9F        STKVG          10
09          BF        INCG
0A          93        STKV4          20
0C          B3        INC4
0D          08        CLRPT

INCOMING  15:17:55.6  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 3B  ROUTE= 0080  ERROR= 00  LENGTH=2E
0A 01  A6 10  35  35  54  00  0C  00  00  5B  00  00  00  00  00  00  00  00  00  00
00  00  00 3E 01  02  0F  00  38  03  35  00  00  00  00  00
```

```
C) OUTGOING  15:17:56.9  NODE TYPE= TM_NODE
NN= 001F  TN= 0001  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=12
AGENT= CKT                          OTMF2    1
81  0A  02  06  81  0E  00  55  BD  08

00          81      STKI2          0A 02
03          06      OPMRPT
04          81      STKI2          0E 00
07          55      STKRX
08          BD      INCE
09          08      CLRPT

INCOMING  15:17:56.9  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 3B  ROUTE= 0080  ERROR= 00  LENGTH=18
0A  02  A3  A4  8D  8C  02  33  00  07  E0  90  4E  CD  89  32

D) OUTGOING  15:17:58.1  NODE TYPE= TM_NODE
NN= 001F  TN= 0001  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=10
AGENT= CKT                  OTMF2    1
81  0A  03  06  50  51  B1  08

00          81      STKI2          0A 03
03          06      OPMRPT
04          50      GETSC
05          51      GETXSC
06          B1      INC2
07          08      CLRPT

INCOMING  15:17:58.2  NODE TYPE= TM_NODE   TEST_ACK_MSG
NN= 001F  TN= 0000  MSGTAG= 3B  ROUTE= 0080  ERROR= 00  LENGTH=0C
0A  03  00  00
END OF INTERCEPTED I/O MESSAGES.
PMIST MULTI USER:
```

Examples

See above example.

Responses

See above example.

qpup

Function

Use the qpup command to print all available PUPI commands and their syntax.

qpup command parameters and variables

Command	Parameters and variables
qpup	There are no parameters or variables.

Qualifications

None

Examples

None available

Responses

None available

ref

Function

Use the ref command to cause the peripheral processor to send a report containing the 14 bytes of reflex buffer accessed by the REFLEX and STKRX primitives for the specified terminal.

ref command parameters and variables

Command	Parameters and variables	
ref	*node*	*term*

Parameters and variables	Description
node	Specifies the PM node number
term	Specifies the PM terminal number

Qualifications

Sample session with the REF command

Following is an example of the output produced by the ref command.

The include command allows PMIST to display the outgoing message produced by the ref command. The include command is not a requirement for the ref command.

```
PMIST MULTI USER:
INTERCEPT BOTH MON
PMIST MULTI USER:
TM8NT 3 0
NODENO=31
TRMNL=
1

PMIST MULTI USER:
INCLUDE 31 1
PMIST MULTI USER:
CMSGLIST
PMIST MULTI USER:
REF 31 1
PMIST MULTI USER:
DMSGLIST

OUTGOING  15:12:49.6  NODE TYPE= TM_NODE
NN= 001F  TN= 0001  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=12
AGENT= CKT                OTMF2    1
```

ref (end)

```
81  0A  02  06  81  0E  00  55  BD  08

00          81      STKI2           0A 02
03          06      OPMRPT
04          81      STKI2           0E 00
07          55      STKRX
08          BD      INCE
09          08      CLRPT

INCOMING  15:12:49.7  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 3B  ROUTE= 0080  ERROR= 00  LENGTH=18
0A 02 A3 A4 8D 8C 02 33 00 07 E0 90 4E CD 89 32
END OF INTERCEPTED I/O MESSAGES.
PMIST MULTI USER:
```

In the previous example, the outgoing message contains the instructions sent to the TM8 by the ref command.

The first STKI2 bytes stack the two bytes (0A 02) required by the OPMRPT primitive. These two bytes are the message type (0A 02).

The second STKI2 primitive stacks the parameters required by the STKRX primitive. #0E represents the number of bytes to be stacked, and #00 represents the offset into the reflex buffer. Therefore, STKRX stacks #0E bytes starting at offset #00 in the reflex buffer.

The INCE primitive includes #0E bytes from the stack in the open report.

The CLRPT closes the report and queues it for output.

The incoming message is sent from terminal 0 and contains the message type (0A 02) and 14 bytes of the reflex buffer accessed by the STKRX and REFLEX primitives.

Examples

See above example.

Responses

See above example.

scpts

Function

Use the scpts command to cause the peripheral processor to send in a report containing the state of the 8-bit (or 16-bit) interface to the peripheral (for example, tip and ring).

scpts command parameters and variables

Command	Parameters and variables	
scpts	*node*	*term*

Parameters and variables	Description
node	Specifies the PM node number
term	Specifies the PM terminal number

Qualifications

Using the scpts command on a TM will give the true condition of the scan points and the signal distribution points. Using the scpts command on a DCM will only give the scan points.

Sample session with the scpts command

Following is an example of the output produced by the scpts command.

The include command allows PMIST to display the outgoing message produced by the scpts command. The include command is not a requirement for the scpts command.

```
PMIST MULTI USER:
INTERCEPT BOTH MON
PMIST MULTI USER:
TM8NT 3 0
NODENO=31
TRMNL=
1
```

scpts (continued)

```
PMIST MULTI USER:
INCLUDE 31 1
PMIST MULTI USER:
CMSGLIST
PMIST MULTI USER:
SCPTS 31 1
PMIST MULTI USER:
DMSGLIST

OUTGOING  15:13:06.2  NODE TYPE= TM_NODE
NN= 001F  TN= 0001  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=10
AGENT= CKT                OTMF2    1
81  0A  03  06  50  51  B1  08

00          81        STKI2          0A 03
03          06        OPMRPT
04          50        GETSC
05          51        GETXSC
06          B1        INC2
07          08        CLRPT

INCOMING  15:13:06.2  NODE TYPE= TM_NODE   TEST_ACK_MSG
NN= 001F  TN= 0000  MSGTAG= 3B  ROUTE= 0080  ERROR= 00  LENGTH=0C
0A  03  00  00
END OF INTERCEPTED I/O MESSAGES.
PMIST MULTI USER:
```

In the previous example, the outgoing message contains the instructions sent to the TM8 by the scpts command.

The STKI2 primitive stacks the two bytes (0A 03) required by the OPMRPT primitive. These two bytes are the message type.

The GETSC primitive stacks the 8 bit interface to the terminal. This is the terminal scan data. The GETXSC stacks the additional 8 bit interface. This is the extended scan data.

The INC2 primitive includes the 2 bytes of stack in the open report.

The CLRPT primitive closes the report and queues it for output.

The incoming message is sent from terminal 0 and contains the message type (0A 03) as well as the scan byte (00) and the extended scan byte (00), respectively.

Examples

None available

Responses

None available

stop

Function

Use the stop command to read the contents of the trace buffer. The stop command creates an incoming maintenance report on terminal 0 of the specified node, which includes a Start Up Message, the terminal number being traced, the index into the trace buffer of the last exec identifier written, and the 32 exec identifiers in the buffer.

stop command parameters and variables

Command	Parameters and variables	
stop	*node*	*buffer*

Parameters and variables	Description
buffer	Specifies the trace buffer number. Valid entries are 0 or 1.
node	Specifies the node number.

Qualifications

The stop command does not terminate the tr command. Instead, it dumps the contents of the trace buffer.

The stop and tr commands only work on the old PM, such as LM, TM, DCM, and MTM.

Use buffer 0 with LM, RLM, and RSM.

Sample session with the stop command

Following is an example of the output produced by the stop command.

The include command allows PMIST to display the outgoing message produced by the stop command. The include command is not a requirement for the stop command.

```
PMIST MULTI USER:
INTERCEPT BOTH MON
PMIST MULTI USER:
TM8NT 3 0
NODENO=31
TRMNL=
1
```

stop (continued)

```
PMIST MULTI USER:
INCLUDE 31 1
PMIST MULTI USER:
CMSGLIST
PMIST MULTI USER:
TR 31 1 0
PMIST MULTI USER:
DMSGLIST

OUTGOING  15:13:31.1  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=0C
81   00  01  62

00          81       STKI2          00 01
03          62       TRAEX

END OF INTERCEPTED I/O MESSAGES.
PMIST MULTI USER:
CMSGLIST
PMIST MULTI USER:
STOP 31 0
PMIST MULTI USER:
DMSGLIST

OUTGOING  15:15:24.4  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=10
81  0A  00  05  80  00  63  08

00          81       STKI2          0A 00
03          05       OPRPT
04          80       STKI1          00
06          63       TRARP
07          08       CLRPT

INCOMING  15:15:24.4  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 3B  ROUTE= 0080  ERROR= 00  LENGTH=2C
0A 00  01  17  44  06  4B  37  8D  3A  A4  32  A5 4E  A3  32  44  06  4B  37  8D 3A
A4 32  A5 4E A3  32  FF  FF  FF  FF  FF  FF  FF FF
END OF INTERCEPTED I/O MESSAGES.
PMIST MULTI USER:
```

In the previous example, the first outgoing message contains the instructions sent to terminal 0 of the TM8 by the tr command.

The STKI2 command stacks the buffer number (00) and terminal number (01) to be used as parameters by TRAEX.

The TRAEX primitive starts storing exec identifiers into the specified buffer.

The second outgoing message contains the instructions sent to terminal 0 of the TM8 by the stop command.

The STKI2 primitive stacks the two bytes (0A 00) required by the OPRPT primitive. These bytes are the message type.

The OPRPT primitive selects a report buffer and initializes it to 0.

The STKI1 primitive stacks the buffer number (00) required by the TRARP primitive. The TRARP primitive includes the specified trace buffer in the open report.

The CLRPT primitive closes the report and queues it for output.

The incoming message is sent from terminal 0 and contains the message type (0A 00). The byte following the message type (01) indicates the location of the last byte written. The next byte (17) is the index indicating where the last exec identifier was written. Since the index byte is #17 in this example, count #17 bytes from the index byte to identify the last exec identifier written.

The message body contains the exec identifiers contained in the trace buffer.

Examples

See above example.

Responses

See above example.

tm8nt

Function

Use the tm8nt command to calculate the node and terminal numbers for TM8.

tm8nt command parameters and variables

Command	Parameters and variables	
tm8nt	*tm8_no*	*ckt*

Parameters and variables	Description
ckt	Specifies the TM8 circuit number
tm8_no	Specifies the identification number of the TM8.

Qualifications

None

Examples

None available

Responses

None available

Function

Use the tr command to initialize the trace buffer and start an exec trace. The trace buffer is dumped with the stop command.

tr command parameters and variables

Command	Parameters and variables		
tr	*node*	*term*	*buffer*

Parameters and variables	Description
buffer	Specifies the trace buffer. Valid entries are 0 or 1.
node	Specifies the node number.
term	Specifies the terminal number.

Qualifications

The tr command cannot be terminated unless the PM is reloaded.

The tr and stop commands only work on the old PM, such as LM, TM, DCM, and MTM.

Use buffer 0 with LM, RLM, and RSM.

The exec trace utility only takes one byte (0-255) for a terminal number. However, Line Modules (LM) and Remote Line Modules (RLM) have more than 256 terminals. Therefore, it is possible to have more than one terminal being traced. For example, if an exec trace is specified for terminal 1 on an LM, terminals 257 and 513 will also be traced, and the trace results will be inaccurate.

To verify that the terminals that could be traced (such as terminals 257 and 513 in the previous example) are idle, use PMIST on the terminals that could have been involved in the trace to determine if the terminals were active. Another method is to enter the LTP access level and post the lines that could have been involved in the trace to determine if they are busy during the trace.

Sample session with the tr command

Following is an example of the output produced by the tr command.The include command allows PMIST to display the outgoing message produced by the tr command. The include command is not a requirement for the tr command.

tr (continued)

```
PMIST MULTI USER:
INTERCEPT BOTH MON
PMIST MULTI USER:
TM8NT 3 0
NODENO=31
TRMNL=
1

PMIST MULTI USER:
INCLUDE 31 1
PMIST MULTI USER:
CMSGLIST
PMIST MULTI USER:
TR 31 1 0
PMIST MULTI USER:
DMSGLIST

OUTGOING  15:13:31.1  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=0C
81   00  01  62

00          81       STKI2          00 01
03          62       TRAEX

END OF INTERCEPTED I/O MESSAGES.
PMIST MULTI USER:
CMSGLIST
PMIST MULTI USER:
STOP 31 0
PMIST MULTI USER:
DMSGLIST

OUTGOING  15:15:24.4  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=10
81  0A  00  05  80  00  63  08

00          81       STKI2          0A 00
03          05       OPRPT
04          80       STKI1          00
06          63       TRARP
07          08       CLRPT

INCOMING  15:15:24.4  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 3B  ROUTE= 0080  ERROR= 00  LENGTH=2C
0A 00  01  17  44  06  4B  37  8D  3A  A4  32  A5 4E  A3  32  44  06  4B  37  8D 3A
```

```
A4 32  A5 4E A3  32  FF  FF  FF  FF  FF  FF  FF FF
END OF INTERCEPTED I/O MESSAGES.
PMIST MULTI USER:
```

In the previous example, the first outgoing message contains the instructions sent to terminal 0 of the TM8 by the tr command.

The STKI2 command stacks the buffer number (00) and terminal number (01) to be used as parameters by TRAEX.

The TRAEX primitive starts storing exec identifiers into the specified buffer.

The second outgoing message contains the instructions sent to terminal 0 of the TM8 by the stop command.

The STKI2 primitive stacks the two bytes (0A 00) required by the OPRPT primitive. These bytes are the message type.

The OPRPT primitive selects a report buffer and initializes it to 0.

The STKI1 primitive stacks the buffer number (00) required by the TRARP primitive. The TRARP primitive includes the specified trace buffer in the open report.

The CLRPT primitive closes the report and queues it for output.

The incoming message is sent from terminal 0 and contains the message type (0A 00). The byte following the message type (01) indicates the location of the last byte written. The next byte (17) is the index indicating where the last exec identifier was written. Since the index byte is #17 in this example, count #17 bytes from the index byte to identify the last exec identifier written.

The message body contains the exec identifiers contained in the trace buffer.

Examples

None available

Responses

None available

var

Function

Use the var command to cause the peripheral processor to send a report containing the variable area accessed by the POPVn and STKVn primitives for the specified terminal. The report will contain the variable area and the time the message was received by the CC.

var command parameters and variables

Command	Parameters and variables	
var	*node*	*term*

Parameters and variables	Description
node	Specifies the PM node number
term	Specifies the PM terminal number

Qualifications

Sample session with the var command

Following is an example of the output produced by the var command.

The include command allows PMIST to display the outgoing message produced by the var command. The include command is not a requirement for the var command.

```
PMIST MULTI USER:
INTERCEPT BOTH MON
PMIST MULTI USER:
TM8NT 3 0
NODENO=31
TRMNL=
1

PMIST MULTI USER:
INCLUDE 31 1
PMIST MULTI USER:
CMSGLIST
PMIST MULTI USER:
VAR 31 1
PMIST MULTI USER:
DMSGLIST

OUTGOING  15:12:35.6  NODE TYPE= TM_NODE
NN= 001F  TN= 0001  MSGTAG= 00  ROUTE= 0584  ERROR= 00  LENGTH=16
```

var (continued)

```
AGENT= CKT                OTMF2    1
81  0A  01  06  9F  00  BF  9F  10  BF  93  20  B3  08

00          81        STKI2          0A 01
03          06        OPMRPT
04          9F        STKVG          00
06          BF        INCG
07          9F        STKVG          10
09          BF        INCG
0A          93        STKV4          20
0C          B3        INC4
0D          08        CLRPT

INCOMING  15:12:35.6  NODE TYPE= TM_NODE
NN= 001F  TN= 0000  MSGTAG= 3B  ROUTE= 0080  ERROR= 00  LENGTH=2E
0A 01  A6 10  35  35  54  00  0C  00  00  5B  00  00  00  00  00  00  00  00  00  00
00  00  00 3E 01  02  0F  00  38  03  35  00  00  00  00  00
END OF INTERCEPTED I/O MESSAGES.
PMIST MULTI USER:
```

In the previous example, the outgoing message contains the instructions sent to the TM8 by the var command.

The STKI2 primitive stacks the two bytes (0A 01) required by the OPMRPT primitive. These two bytes are the message type. The OPMRPT primitive selects a maintenance report buffer and initializes it to 0.

The first STKVG primitive stacks 16 bytes of variable area starting at offset #00 in the variable area. The first INCG primitive includes the 16 bytes of stack in the open report.

The second STKVG primitive stacks 16 bytes of variable area starting at offset #10 in the variable area. The second INCG primitive includes the 16 bytes of stack in the open report.

The STKV4 primitive stacks 4 bytes of variable area starting at offset #20 in the variable area. The INC4 primitive includes the 4 bytes of stack in the open report.

The CLRPT primitive closes the open report and queues it for output.

The incoming message is sent from terminal 0 and contains the message type (0A 01). The remaining bytes of the message are the bytes from the terminal variable area.

Examples

None available

Responses

None available

Primitives

Outgoing messages displayed in PMIST output contain a list of primitives called by the CC.

This chapter contains a list of commonly used primitives.

For more information on primitives, refer to module PPCD in the PROTEL listings.

Primitive organization

Primitives are organized as follows:

- Operation primitives (00 - 7F) perform arithmetic, logical, and terminal process functions.
- Stack primitives (80 - FF) perform stack manipulation functions and include in their opcodes the number of parameters used by the primitive.

Primitive definitions

The description of each primitive includes its explanation using the following format:

OPCODE o1 - on !! n1 - nn ! q1 - qn .

The symbols !! and ! are used as delimiters between the operands and parameter types. A delimiter is a character that marks the beginning or the end of a unit of data. An operand is an entity to which an operation is applied.

In the above format example:

- o1 - on are operands. These are values that follow the opcode in the primitive string.
- n1 - nn are the parameters that are POPped from the parameter stack by the primitive. The n1 parameter is on the top of the stack.
- q1 - qn are parameters that are PUSHed on the stack by the primitive. The q1 parameter is on top of the stack.

Following are the primitive definitions. Each heading supplies the primitive opcode on the left and the primitive name on the right. Following each

heading is a description of how the primitive operates, the format of the primitive, and an explanation of the primitive parameters.

00 NOOP

NOOP does nothing

01 STOP

STOP resets the reflex flag to indicate that instructions from the CC have been completed or an exec has been processed, and begins processing the reflex buffer if a reflex message is pending. If no reflex buffer exists, the primitive releases the open command buffer, stops the server timer, clears the inject request for utility message, unblocks the terminal, and ends the command processing sequence without closing any open reports.

This primitive is usually the last primitive of a command sequence.

The format for STOP is as follows:

```
STOP
```

02 SETMTC

SETMTC sets the PP maintenance register, which includes the 'under test' lamp.

The format for SETMTC is as follows:

```
SETMTC !! mask  final_state
```

Where:

mask is the parameter(s) on which comparative decisions are made.

final_state is the state of the peripheral processor maintenance register.

03 SIGNL

SIGNL signals the active terminal process that an event, such as an enqueued exec, has occurred.

This primitive is used in conjunction with such terminal processes as digit reception or digit outpulsing.

The format for SIGNL is as follows:

```
SIGNL !! mp
```

Where:

mp is the terminal process or minor process identifier. A value of #00 specifies terminal process A. A value of #01 specifies terminal process B.

04 RSTTRML

RSTTRML idles both terminal processes and the background integrity and timer sequences, unblocks the terminal, disassociates the channel, and clears the integrity checks. RSTTRML also disassociates the channel, sets the terminal status to NIL_TSI, sets the reflex message type to zero, and sets the Signal Distributor (SD) points to on-hook.

If terminal 0 is used, no action is taken.

The format for RSTTRML is as follows:

RSTTRML

05 OPRPT

OPRPT attempts to select a report buffer for 40 ms or until the server timer times out. If it is not successful, OPRPT sets the terminal status indicator to NIL_TSI. If it is successful, OPRPT sets up the report header and saves the report pointer.

The two bytes of the message type are popped out from the parameter stack and placed after the header of the report.

The format for OPRPT is as follows:

OPRPT !! msg_type_1 msg_type_0

Where:

msg_type_1 is the second byte of the two-byte message type (the second byte after the message header in the report).

msg_type_0 is the first byte of the two-byte message type (the first byte after the message header in the report).

06 OPMRPT

OPMRPT opens a maintenance report and sets the terminal identifier to zero.

The format for OPMRPT is as follows:

OPMRPT !! msg_type_1 msg_type_0

Where:

msg_type_1 is the second byte of the two-byte message type (the second byte after the message header in the report).

msg_type_0 is the first byte of the two-byte message type (the first byte after the message header in the report).

07 OPLRPT

OPLRPT opens a low priority report. If in normal priority mode, OPLRPT opens a report (OPRPT). If in high priority or overload mode, or if the report buffer is not available, OPLRPT processes the conditional clause.

The format for OPLRPT is as follows:

OPMRPT branch_value cond_clause !! msg_type_1 msg_type_0

Where:

branch_value is the number of bytes, including itself, needed to skip over the conditional clause.

cond_clause is the command sequence to process if the I/O limitations are exceeded.

msg_type_1 is the second byte of the two-byte message type (the second byte after the message header in the report).

msg_type_0 is the first byte of the two-byte message type (the first byte after the message header in the report).

08 CLRPT

If a report is open and the message type corresponds to the reflex message type, CLRPT activates the reflex message; otherwise, the report is queued for output. If no report is open, an error (#9) is generated.

The report length is limited to 64 bytes, including the 8-byte message header.

CLRPT will fail if the report length (including the header) is less than 10 bytes or if the report choke limit is exceeded.

The format for CLRPT is as follows:

CLRPT

09 REFTRM

REFTRM sets the indicated terminal as the reference terminal. If the terminal identifier is out of range, an error is indicated.

The format for REFTRM on a TM and a DCM is as follows:

REFTRM !! term_id

The format for REFTRM on a LM and RLM is as follows:

REFTRM !! termid_msb termid_lsb

Where:

termin_id is a terminal identifier, and can be one of the following:
TM 0-31 (0-#1F)
DCM 0-120 (0-#78)
LM 0-640 (0-#280)
RLM 0-640 (0-#280) non-ESA mode
0-1280 (0-#500) ESA mode

termid_msb is the terminal identifier's most significant byte.

termid_lsb is the terminal identifier's least significant byte.

0A MRPHDR

If the record is open, the data byte from the top of the parameter stack is written into the report (message) header at the specified offset.

If the offset is greater than 9 or no report is open, an error is indicated.

The format for MRPHDR is as follows:

MRPHDR offset !! data

Where:

offset is the byte offset into the report header (#00-#09).

data is one byte used to modify the report header.

0B CPYMSG

If a report is open, the current message is copied into the report buffer. If no report is open, an error is indicated.

The format for CPYMSG is as follows:

CPYMSG

0C NVKOP

NVKOP processes the command on the top of the stack.

NVKOP !! opcode (parm1...parmn)

Where:

opcode is the opcode of a primitive intended for processing.

parm1...parmn are the parameters required by the invoked primitive.

0D KILLTMR

For a DCM and TM, KILLTMR stops the background timer and resets any pending time-out process on the reference terminal.

For an LM and RLM, KILLTMR stops both the short and the long background timers on the referenced channel and removes enqueued time-out messages.

The format for KILLTMR is as follows:

```
KILLTMR
```

0E STIME

STIME sets the 10 ms background timer and sets it for t periods. When the background timer times out, the specified exec is posted.

The format for STIME is as follows:

```
STIME !! exec_id t
```

Where:

exec_id is the exec identifier of the exec that will be posted when the background timer times out.

t is the number of 10 ms periods to wait before the background timer times out.

For LM and RLM, STIME idles any currently running STIME on the specific channel. STIME and LTIME may run concurrently.

For DCM and TM, STIME idles any currently running STIME or LTIME on the terminal. A KILLTMR should precede an STIME to prevent difficulties in clearing the timer under heavy traffic.

For a DCM, parameter t is decremented by 1 before the timer is started.

0F LTIME

For DCM and TM, LTIME turns on the 160 ms background timer and sets the timing value for t periods. When the background timer times out, the specified exec is posted.

For LM and RLM, LTIME turns on the 10 ms background timer and sets the timing value for t periods, where t is a 2-byte timing value. When the background timer times out, the specified exec is posted.

The format for LTIME for DCMs and TMs is as follows:

```
LTIME !! exec_id t
```

Where:

exec_id is the exec identifier of the exec that will be posted when the background timer times out.

t is the number of 10 ms periods to wait before the background timer times out.

The format for LTIME for LMs and RLMs is as follows:

LTIME !! exec_id t_msby t_lsby

Where:

exec_id is the exec identifier of the exec that will be posted when the background timer times out.

t_msby is the most significant byte of the timing value; specifying the number of 10 ms periods to wait before the background timer times out.

t_lsby is the least significant byte of the timing value; specifying the number of 10 ms periods to wait before the background timer times out.

For DCM and TM, timer precision is within 160 ms. Therefore, timeout occurs between ((T-1)*160)ms and (T*160)ms. The following is also true:

- Starting LTIME on a terminal will idle any LTIMER or STIMER currently running on that terminal. However, a KILLTMR should be processed before starting LTIME because the DCM may have difficulties clearing a timer if a new one is requested without explicitly killing the old one under heavy traffic.
- LTIMER and STIMER are not independent; therefore, only one may be active at any given time. (t = 00 -> 256 * 160 ms)
- For DCMs, parameter t is decremented by 1 before the timer is started.

For LM and RLM, starting LTIME on a channel will idle any LTIMER running on that channel. LTIMER is independent of STIMER allowing both timers to be active at the same time. The two-byte timing value allows a timing range of 0 milliseconds to (256*256*10) milliseconds. (t = 0000 -> 0 to 10 ms)

10 IFEQ

IFEQ processes the conditional clause if N1 = N2.

The format for IFEQ is as follows:

IFEQ branch_value cond_clause !! n1 n2

Where:

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause).

cond_clause is the conditional clause containing the command sequence processed if the conditions are met.

Note: The conditional clause is true if N2 - N1 = 0.

n1 is one byte of data for testing.

n2 is one byte of data for testing.

11 IFNE

IFNE processes the conditional clause if N1 <> N2.

The format for IFNE is as follows:

IFNE branch_value cond_clause !! n1 n2

Where:

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause).

cond_clause is the conditional clause containing the command sequence processed if the conditions are met.

Note: The conditional clause is true if N2 - N1 <> 0.

n1 is one byte of data for testing.

n2 is one byte of data for testing.

12 IFGE

IFGE processes the conditional clause if N1 >= N2.

The format for IFGE is as follows:

IFGE branch_value cond_clause !! n1 n2

Where:

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause).

cond_clause is the conditional clause containing the command sequence processed if the conditions are met.

Note: The conditional clause is true if N2 - N1 <= 0.

n1 is one byte of data for testing.

n2 is one byte of data for testing.

13 IFLE

IFLE processes the conditional clause if N1 <= N2.

The format for IFLE is as follows:

IFLE branch_value cond_clause !! n1 n2

Where:

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause).

cond_clause is the conditional clause containing the command sequence processed if the conditions are met.

Note: The conditional clause is true if N2 - N1 <= 0.

n1 is one byte of data for testing.

n2 is one byte of data for testing.

14 IFGT

IFGT processes the conditional clause if N1 > N2.

The format for IFGT is as follows:

IFGT branch_value cond_clause !! n1 n2

Where:

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause).

cond_clause is the conditional clause containing the command sequence processed if the conditions are met.

Note: The conditional clause is true if N2 - N1 < 0.

n1 is one byte of data for testing.

n2 is one byte of data for testing.

15 IFLT

IFLT processes the conditional clause if N1 < N2.

The format for IFLT is as follows:

IFLT branch_value cond_clause !! n1 n2

Where:

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause).

cond_clause is the conditional clause containing the command sequence processed if the conditions are met.

Note: The conditional clause is true if N2 - N1 > 0.

n1 is one byte of data for testing.

n2 is one byte of data for testing.

16 KLOP0

KLOP0 idles terminal process A and clears any execs posted by terminal process A.

KLOP0 should be processed prior to starting terminal process A to prevent clean-up difficulties during heavy traffic.

The format for KLOP0 is as follows:

KLOP0

17 KLOP1

KLOP1 idles terminal process B and clears any execs posted by terminal process B.

KLOP1 should be processed prior to starting terminal process A to prevent clean up difficulties during heavy traffic.

The format for KLOP1 is as follows:

KLOP1

18 EXTXEC

EXTXEC ensures that an exec exists. This primitive adds the exec text data to the most recently defined exec, send an acknowledgment, and stops processing further commands. An exec defined by DEFXEC must be

extended by EXTXEC before any other execs are defined. For information on DEFXEC, refer to opcode 24 DEFXEC on page 7-15.

An error message is generated for an invalid branch value or when the exec store overflows.

The format for EXTXEC is as follows:

EXTXEC branch_value data1...datan

Where:

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause).

data1...datan is the command sequence to be added to an exec.

19 REFLEX

REFLEX loads the reflex message into the terminal reflex message buffer and puts a STOP code at the end of the message. On a CLRPT, the message type in the message being closed is checked against the reflex message type. If they match, the reflex will be invoked.

An error is generated if the reflex message is too long or if terminal 0 is used.

The format for REFLEX is as follows:

REFLEX msg_type_0 msg_type_1 branch_value byte1...byten

Where:

msg_type_0 is the first byte of the two-byte message type (the first byte after the message header in the report).

msg_type_1 is the second byte of the two-byte message type (the second byte after the message header in the report).

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause). The range is #02 - #0F.

byte1...byten specifies one to 14 bytes of reflex message data.

1A XMNTG

XMNTG removes the integrity value from the parameter stack and transmits it in the Channel Supervision Message (CSM).

For the TM, XMNTG fails if no channel has been associated.

An error message is generated if no channel has been associated or if terminal 0 is used.

The format for XMNTG is as follows:

XMNTG !! integrity
Where:

integrity is the integrity value intended for transmission in the CSM.

1B RDDAT

RDDAT pushes the addressed data byte onto the parameter stack.

The format for RDDAT is as follows:

RDDAT !! msbyad lsbyad ! data
Where:

msbyad is the most significant byte of the address from where the data byte will be read.

lsybyad is the least significant byte of the address from where the data byte will be read.

data is the data byte.

1C WRDAT

WRDAT writes the data byte into the specified address if the security key (hex value ED) is the correct value. An error is generated if the security key is invalid.

The format for WRDAT is as follows:

WRDAT sec_key !! data msbyad lsbyad
Where:

sec_key is the security key, which is equal to #ED.

data is one byte intended for the specified address.

msbyad is the most significant byte of the address where the data byte is to be written.

lsbyad is the least significant byte of the address where the data byte is to be written.

1D CSMIGET

CSMIGET puts the incoming CSM byte on the parameter stack. An error is generated when the stack overflows.

The format for CSMIGET is as follows:

CSMIGET !! ! data
Where:

data is the data that will be put on the parameter stack.

1E CSMOGET

CSMOGET puts the outgoing CSM byte on the parameter stack. An error is generated when the stack overflows.

The format for CSMOGET is as follows:

CSMOGET !! ! data
Where:

data is the data that will be put on the parameter stack.

1F CASEX

CASEX calls one of four execs based on the conditions that follow:

exec0 IF (value .AND. mask) = 0
exec1 IF (value .AND. mask) < (highest bit set in mask)
exec2 IF (value .AND. mask) = (highest bit set in mask)
exec3 IF (value .AND. mask) > (highest bit set in mask)

> ***Note:*** The highest bit set in the mask is defined as the highest set bit of the mask followed by that bit number minus 1. Therefore, 01001010 has the highest set bit of mask value of 01000000.

An error is generated when an exec that is out of range has been referenced or if an undefined exec has been referenced.

The format for CASEX is as follows:

CASEX !! exec3 exec2 exec1 exec0 mask value
Where:

exec3 is a valid exec sequence identifier (from #00 to #FE).

exec2 is a valid exec sequence identifier (from #00 to #FE).

exec1 is a valid exec sequence identifier (from #00 to #FE).

exec0 is a valid exec sequence identifier (from #00 to #FE).

mask value are the parameters on which comparative decisions are made.

20 CHKNTG

CHKNTG requests a background integrity check on the associated channel. The integrity exec is invoked if integrity is lost or if the process detects a parity problem.

The integrity value at TVA.EXINT is transferred to the network compare memory and a check for mismatch is enabled. The parity failure bit at TVA.TXINT is reset. CDB monitoring for mismatch is enabled.

If an integrity mismatch occurs or if excessive parity errors are detected, the exec at TVA.ITTXI is moved to TVA.INTXI, where it is posted.

Integrity mismatches are filtered for 80 to 160 ms before invoking the integrity_lost_exec.

The format for CHKNTG is as follows:

CHKNTG

21 IGNNTG

IGNNTG cancels any background integrity checks (LGNTG and CHKNTG) for the associated channel and clears any pending integrity exec postings. Incoming CSM is disabled as well.

The format for IGNNTG is as follows:

IGNNTG

22 LFNTG

LFNTG requests a background integrity check on the associated channel. The integrity exec is invoked if integrity is found. If integrity is not found after 240 ms, the integrity time-out exec overwrites the integrity exec and is invoked.

The integrity value at TVA.EXINT is transferred to the network compare memory and a check for match is enabled. The parity failure bit at TVA.TXINT is reset, and CDB monitoring is disabled.

If an integrity match occurs, the integrity exec at TVA.ITTXI is moved to TVA.INTXI where it is posted. If after 160-240 ms an integrity match is not found, the integrity time-out exec at TVA.INTXI is posted.

Integrity matches are filtered for 80 to 160 ms before invoking the integrity_found_exec.

The format for LFNTG is as follows:

LFNTG

23 XEXEC

XEXEC resets the exec store and all terminals that have a channel associated if the security key is valid. If the reset code is equal to 0, all terminals are reset. An error is generated if the security key is valid or if a terminal other than terminal 0 is used.

For reset terminal procedures, refer to opcode 04 RSTTRML on page 7-3.

The format for XEXEC is as follows:

XEXEC !! control_byte sec_key

Where:

control_byte is one of the following:

- **0** clears all terminals and exec store.
- **1** clears exec store only.
- **2** resets the C-side channels (0-255) given by the bit map.
- **3** resets the C-side channels (256-571) given by the bit map.

sec_key is the security key, which is equal to #ED.

24 DEFXEC

DEFXEC ensures that the exec identifier has not already been defined and that the exec will not overflow the exec table. DEFXEC copies the exec into the exec table and the data is entered into exec store. The exec consists of a branch value, which is one byte, followed by an ENDXEC opcode. An acknowledgement report is generated and command processing continues.

If more than one exec is defined in a message, only one acknowledgement is sent. Commands other than DEFXEC or REDXEC cause termination of command string processing.

Errors are generated for an invalid branch value, invalid exec_id, exec store overflow, or an attempt to define execs if exec store is write-protected.

The format for DEFXEC is as follows:

DEFXEC exec_id branch_value data1...datan

Where:

exec_id is the exec identifier, from #01 to #FE.

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause). A branch value of 0 is invalid.

data1 is the first primitive command of an exec sequence.

datan is the last primitive command of an exec sequence.

25 ENDXEC

ENDXEC checks for stack underflow and pops the return command pointer and the previous parameter stack base from the stack. The parameter stack pointer does not need to be pointing to the return address because ENDXEC locates the last return address entered on the stack. However, all data on the stack above the return address is lost.

All the parameters needed by this primitive are automatically entered on the parameter stack when an exec is called or invoked.

An error is generated for stack underflow.

The format for ENDXEC is as follows:

ENDXEC !! cdptr_msby cdptr_lsby psb

Where:

cdptr_msby is the most significant byte of the return command pointer.

cdptr_lsby is the least significant byte of the return command pointer.

psb is the previous parameter stack base.

26 NVKXEC

NVKXEC ensures that the exec identifier is valid and pushes the command pointer and parameter stack base pointer onto the stack, putting the exec address in the command pointer.

An error is generated if an exec is outside the valid range or if the exec has not been defined.

The format for NVKXEC is as follows:

NVKXEC !! exec_id ! cdptr_msby cdptr_lsby psb

Where:

exec_id is the exec sequence identifier, from #00 to #FE.

cdptr_msby is the most significant byte of the return command pointer.

cdptr_lsby is the least significant byte of the return command pointer.

psb is the previous parameter stack base.

27 NXTDIG

NXTDIG is used in conjunction with the outpulsing primitives (MF or DP) to initiate the sending of the next outgoing digit. This opcode must be processed before each digit is sent.

For specific usage examples, refer to opcode 5D S2DPOTP on page 7-47 and opcode 5F S2MFOTP on page 7-51.

The format for NXTDIG is as follows:

NXTDIG

28 CONPCM

CONPCM connects the terminal Pulse Code Modulation (PCM) to the associated channel and transmits on both ports.

The format for CONPCM is as follows:

CONPCM

29 DISPCM

DISPCM sets the terminal PCM to idle and stops transmitting speech.

The format for DISPCM is as follows:

DISPCM

2A SWCHPCM

SWCHPCM selects PCM input from the other network plane and toggles the current plane bit (most significant bit) of PB.TXINT.

The format for SWCHPCM is as follows:

SWCHPCM

2B CONTN

CONTN connects the specified tone to the terminal and channel and disables speech transmission.

The format for CONTN is as follows:

CONTN !! tone_id

Where:

tone_id is the PCM tone identifier. Refer to Table 7-1 for a list of tone identifiers.

Table 7-1xxx
PCM tone identifiers

Tone code	Tone name
00	PCM
01	
02	
03	
04	AR - audible ringing
05	BT - busy tone
06	DT - dial tone
07	
08	
09	HT - high tone
0A - 0E	
0F	400 Hz dial tone
10	IT - idle tone
11	MF1
12	MF2 (coin collect)
13	MF3
14	MF4
15	MF5
16	MF6
17	MF7
18	MF8
19	MF9
1A	MF0
1B	MF11 (ST3P & ringback)
1C	MF12 (STP)
1D	MF - KP (coin return & KP1)
1E	MF - KP2 (ST2P)
1F	MF - ST

-continued-

Table 7-1xxx
PCM tone identifiers

Tone code	Tone name
20	DTMF tone D
21	DTMF tone 1
22	DTMF tone 2
23	DTMF tone 3
24	DTMF tone 4
25	DTMF tone 5
26	DTMF tone 6
27	DTMF tone 7
28	DTMF tone 8
29	DTMF tone 9
2A	DTMF tone 0
2B	DTMF tone *
2C	DTMF tone #
2D	DTMF tone A
2E	DTMF tone B
2F	DTMF tone C

Note: If the tone identity 00 is specified, PCM will be connected, establishing speech transmission.
Note: For TM, an error is generated if terminal 0 is referenced or if a channel has not been associated.
Note: For LM and RLM, if one of the four valid tone identifiers (#00, #04, #05, and #06) are not specified, then quiet tone is connected.
Note: An error is generated if an invalid channel identifier is used.

End

2C CONTSS

Note: This primitive is not valid for TMs and DCMs.

CONTSS connects the channel through the time-space switch.

The format for CONTSS is as follows:

CONTSS !! source ch_no
Where:

source ch_no is the channel number.

2D ASSCH

ASSCH associates the given channel with the referenced terminal, sets the PCM type to idle, disables speech transmission, sets the reception to an even port, and initializes the tone levels.

For a DCM, the channel number is not selectable. The terminal will always associate with its corresponding channel(for example, terminal z is associated with channel z). A parameter is required on the stack.

The format for ASSCH is as follows:

ASSCH !! ch_no

Where:

ch_no is the channel intended for association to a terminal.

2E DISCHN

DISCHN idles and releases the associated channel from the referenced terminal and disables speech transmission on the channel.

The format for DISCHN is as follows:

DISCHN

2F SETPAD

In the TM, SETPAD provides digital pad control by allowing setting and resetting of the transmit and receive pad values independently of each other. An error is generated if reference terminal 0 is specified or if a channel has not been associated.

In the DCM, SETPAD does nothing. Also, the value of bits 0 through 7 does not matter.

The format for SETPAD is as follows:

SETPAD !! function

Where:

function provides pad selection, enable/disable, and pad value control. For a TM, the bit assignments are as follows:

bits 0-2 pad value if bit 3 is set

bit 3 can have one of the following values:
- **0** disable selected pad
- **1** enable selected pad

bits 4-6

bit 7 can have one of the following values:
0 select RPAD
1 select XPAD

Refer to Table 7-2 for the receive and transmit values and the corresponding pad settings.

Table 7-2xxx
Receive and transmit values

Receive value	Receive loss pad setting (RPAD)	Transmit value	Transmit gain pad setting (XPAD)
0	0.25	0	0.25
1	0.50	1	0.50
2	0.75	2	.75
3	1.00	3	1.00
4	1.25	4	1.25
5	1.50	5	1.50
6	1.75	6	1.75
7	2.00	7	2.00

End

30 TSTRPA

TSTRPA tests the PP-CC interface to determine whether a report buffer is available and whether it can be sent. If no report is available, or if the report choke limit is reached, the conditional clause is processed.

The format for TSTRPA is as follows:

TSTRPA branch_value conditional_clause

Where:

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause). A branch value of 0 is invalid.

conditional_clause is the command sequence processed if the conditions are met.

31 INCDGR

INCDGR includes the terminal digit register in the open report and updates the number of digits reported to equal the digit count. If bit 0 of the control byte is set, the digit count is updated to the value specified in the most significant nibble of the control byte. An error is generated if no report is open or if the report exceeds the maximum length.

The format for INCDGR is as follows:

INCDGR !! control

Where:

control updates the terminal digit count register if required. Following are the bit assignments:
bit 0 - update the digit count to the value specified in bits 4-7
bit 1-3 - not used.
bit 4-7 - the new digit count value.

32 SCONCHK

SCONCHK processes the conditional clause if the designated scan point identifier is on.

Only one scan point is checked at a time. For LM and LCM, no scan_id parameter is used.

The format for SCONCHK is as follows:

SCONCHK branch_value cond_clause !! scan_id

Where:

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause). A branch value of 0 is invalid.

cond_clause is the command sequence processed if the conditions are met.

scan_id is the scan point identifier. For a DCM, 01 = check the A bit and 02 = check the B bit.

33 SCOFCHK

SCOFCHK processes the conditional clause if the designated scan point identifier is off.

Only one scan point is checked at a time. For LM and LCM, no scan_id parameter is used.

The format for SCONCHK is as follows:

SCONCHK branch_value cond_clause !! scan_id
Where:

branch_value is the number of bytes necessary to skip over the conditional clause (one byte greater than the length of the conditional clause). A branch value of 0 is invalid.

cond_clause is the command sequence processed if the conditions are met.

scan_id is the scan point identifier. For a DCM, 01 = check the A bit and 02 = check the B bit.

34 STSCREAD

Note: This primitive is not valid for TMs and DCMs.

STSCREAD starts terminal process A to perform a read scan.

The format for STSCREAD is as follows:

STSCREAD

35 TIMEDIF4

TIMEDIF4 subtracts two 4-byte numbers.

The format for STSCREAD is as follows:

TIMEDIF4 !! t1t t1h t1m t1l t2t t2h t2m t2l ! t3t t3h t3m t3l
Where:

t1 is the minuend:

t1t higher order byte
t1h high-middle order byte
t1m low-middle order byte
t1l low order byte

Note: In subtraction, the minuend is the number or quantity from which another number or quantity is subtracted.

t2 is the subtrahend:

t2t higher order byte
t2h high-middle order byte
t2m low-middle order byte
t2l low order byte

Note: In subtraction, the subtrahend is the number or quantity subtracted from the minuend.

t3 is the difference:

t3t higher order byte
t3h high-middle order byte
t3m low-middle order byte
t3l low order byte

Note: In subtraction, the difference is the number or quantity that is the result of subtracting the subtrahend from the minuend.

36 SDONSET

SDONSET sets the terminal signal distributor point on.

For the DCM, both the A bit and the B bit may be set simultaneously.

The format for SDONSET is as follows:

SDONSET !! sd_no
Where:

sd_no is the signal distributor point identifier. In the DCM, the sd_no is a bit mask where:
bit 0 if set to 1, operates the A bit.
bit 1 if set to 1, operates the B bit.
bits 2-7 not used.

37 SDOFSET

SDOFSET sets the terminal signal distributor point off.

For the DCM, both the A bit and the B bit may be set simultaneously.

The format for SDOFSET is as follows:

SDOFSET !! sd_no
Where:

sd_no is the signal distributor point identifier. In the DCM, the sd_no is a bit mask where:
bit 0 if set to 1, operates the A bit.
bit 1 if set to 1, operates the B bit.
bits 2-7 are not used.

38 CALLXEC

CALLXEC ensures that the exec identifier is valid and pushes the command pointer and parameter stack base pointer onto the stack, putting the exec address in the command pointer.

An error is generated if an exec is outside the valid range or if the exec is not defined.

The format for CALLXEC is as follows:

CALLXEC exec_id !! ! cdptr_msby cdptr_lsby psb

Where:

exec_id is the exec sequence identifier, from #00 to #FE.

cdptr_msby is the most significant byte of the return command pointer.

cdptr_lsby is the least significant byte of the return command pointer.

psb is the previous parameter stack base.

39 SVONSET

SVONSET turns on the transmission supervision bit.

The format for SVONSET is as follows:

SVONSET

3A SVOFSET

SVOFSET turns off the transmission supervision bit.

The format for SVOFSET is as follows:

SVOFSET

3B WRTG

Note: This primitive is not valid for TMs and DCMs.

WRTG writes into the checksum protected global area.

The format for WRTG is as follows:

WRTG seckey offset !! data

Where:

sec_key is the security key, which is equal to #ED.

offset is the offset from the base of protected global store (00-6F).

data is the data to be written to the protected global area.

3C S3DGREC

Note: This primitive is not valid for TMs and DCMs.

S3DGREC starts terminal process A to perform standard digit collection and pretranslation.

The meaning of bits or nibbles in the parameters are defined by the translation table sent by the CC (excluding the DGT field).

The format for S3DGREC is as follows:

S3DGREC !! dat3 dat2 dat1 dat0

3D SDTREC

SDTREC is an invalid opcode.

3E MAPSV

MAPSV assigns the designated terminal process to map the receive supervision bit into the signal distributor point (sd_no). If the polarity bit is 1, MAPSV changes the polarity when mapping.

For a DCM, the A bit and the B bit can be mapped into simultaneously.

The format for MAPSV is as follows:

MAPSV !! mp sd_no polarity

Where:

mp

mp is the terminal process identifier:
bit 0 0 = terminal process A, 1 = terminal process B
bits 1-7 not used.

sd_no signal distribution point identifier. Following is true for a DCM:
bit 0 set to 1 enables mapping into A bit.
bit 1 set to 1 enables mapping into B bit.
bits 2-7 not used.

polarity is the mapping polarity:
bit 0 0 = same polarity, 1 = reverse polarity
bits 1-7 not used.

3F SVMAP

SVMAP assigns a terminal process to map the scan point scan identifier into the supervision bit.

In the DCM, the A bit and the B bit cannot be mapped into simultaneously.

The format for MAPSV is as follows:

SVMAP !! mp scan_id

Where:

mp is the terminal process identifier:
bit 0 0 = terminal process A, 1 = terminal process B
bits 1-7 not used.

scan_id scan point identifier. Following is true for a DCM:
bit 0 set to 1 enables mapping into A bit.
bit 1 set to 1 enables mapping into B bit.
bits 2-7 not used.

40 STKBIM

STKBIM accumulates a busy/idle map (idle=1, busy=0) of all channels on the same link as the associated channel and puts the status on the stack.

The format for STKBIM is as follows:

STKBIM !! ! bis(24,31) bis(16,23) bis(8,15) bis(0,7)

Where:

bisn is the busy/idle status (1=busy, 0=idle) for channels n (in bit 0) to n+7 (in bit 7).

41 SDTSCAN

SDTSCAN assigns terminal process A to receive a message from the data terminal. The terminal process monitors the data terminal and wait until a message is ready to be sent. The data transfer occurs on the TM extension scan and signal distributor bytes. The message received is stored in the TVA and reflex buffer.

The control byte is sent to the data terminal on the signal distributor byte, unless the control byte is 0 or #FF. If the control byte is #FF, terminal process A will start to receive a message, but the control byte is not sent on the signal distributor byte. If the control byte is 0, terminal process A will not be started, but it will read the status of the process.

When the terminal process is started, it waits for the data terminal to signal that a message is ready to be sent to the TM. This signal is sent on the extension scan byte. The TM replies with SEND and waits 10 ms before

reading the length of the message from the extension scan byte. The TM reads a data byte every 10 ms until the complete message is received.

After the entire message has been received, the terminal process is idled and the data_terminal_report_exec is posted.

The format for SDTSCAN is as follows:

SDTSCAN !! control_byte
Where:

control_byte is a byte that specifies the value that the TM puts on the signal distribution byte. This value is the control byte of the data terminal. The following values are exceptions:
00 signals that the status of the process should be read.
#FF signals that the process should be started without sending a control byte on the signal distribution byte.

While the terminal process is running, the terminal variable area is set up as follows:

```
MPAAR   + 0  =  data_terminal_report_exec_id
MPBAR   + 1  =  data byte 1
MPBAR   + 2  =  data byte 2
   :        :       :    :
MISCB   + 1  =  data byte 16
MISCB   + 2  =  process status
 ---------------------------
REFLEX  + 2  =  data byte 17
REFLEX  + 3  =  data byte 18
   :        :       :    :
REFLEX  + 15=  data byte 30
```

The possible process status in MISCB+2 can be one of:

06 - data terminal transmitting to TM
08 - TM transmitting to data terminal
02 - idle (waiting to send or receive a message)

- The maximum message length is 30 bytes.
- SDTSCAN always starts terminal process A.
- Once the terminal process is started, it will run until a message from the data terminal is received.
- An error is generated if terminal 0 is used (error code #2).

42 CDTB

CDTB copies n data bytes from the command buffer into the terminal process block and starts the data terminal process to transmit the message to the data terminal.

The terminal process waits until the data terminal sends a ready-to-receive message on the extension scan byte. The message is then sent on the extension signal distribution byte. The message is read every 10 ms until the entire message has been received.

After the message is received, the terminal process is restarted to monitor for and receive messages from the data terminals.

The format for CDTB is as follows:

CDTB n data1 data1 ... datan
Where:

n is the number of data bytes in the message.

data1...datan are the data bytes of the message.

CDTB can also take the following format:

CDTB 0 !! n datan ... data2 data1
Where:

0 indicates that the message length and data are on the parameter stack.

Note 1: The maximum message length is 30 bytes. Data over 30 bytes is discarded.

Note 2: The data terminal process should be running before CDTB is used.

Note 3: An error is generated if terminal 0 is used (error code #2), the TV A is already sending or receiving a message (error code #16), or stack underflow occurs (error code 7).

43 TBRP

TBRP copies data from a data terminal buffer into a report. An error byte is contained in the report to indicate whether the received message was greater than 30 bytes or whether the data terminal error flag was set when it sent the message.

After the data is transferred to the report, the data terminal process is restarted to receive a message from the data terminal.

An error is generated if terminal 0 is used (error code #2), the data terminal buffer does not have a message (error code #17), or no report is open (#B).

The report data includes the following:

error_flag byte_count(n) data1 data2 ... datan
where the error count bit assignment is as follows:

bit 1 data byte count overflow (when this bit is set to 1)

bit 5 error bit from data terminal (when this bit is set to 1)

all other bits are not used.

Refer to opcode 41 SDTSCAN on page 7-27 for the terminal buffer layout.

The format for TBRP is as follows:

TBRP

44 ADDPARMS

ADDPARMS adds two values on the parameter stack and returns the sum to the stack.

The format for ADDPARMS is as follows:

ADDPARMS !! n1 n2 ! n3
Where:

n1 is the augend byte

Note: In addition, the addend is a number or quantity that is added to the augend.

n2 is the addend byte

Note: In addition, the augend is a number or quantity to which numbers or quantities are added.

n3 is the sum of n1 and n2.

45 SUBPARMS

SUBPARMS subtracts two values on the parameter stack and returns the difference to the stack.

The format for SUBPARMS is as follows:

SUBPARMS !! n1 n2 ! n3
Where:

n1 is the minuend byte

Note: In subtraction, the minuend is the number or quantity from which another number or quantity is subtracted.

n2 is the subtrahend byte

Note: In subtraction, subtrahend is the number or quantity subtracted from the minuend.

n3 is the difference of n1 and n2.

Note: In subtraction, the difference is the number or quantity that is the result of subtracting the subtrahend from the minuend.

46 SATVR

SATVR starts the signal reception process to monitor and report signals, such as:

- ON HOOK (1)
- OFF HOOK (2)
- FLASH (3)
- PREEMPT TO IDLE (4)
- PREEMPT TO SEIZE (5)
- WINK (6)
- CONFUSION (7).

In the previous list, the number in parentheses is the parameter that is placed at location MP_A+1 to indicate the signal that was detected. The exec in MP_A+0 is invoked after detecting a signal. The primitive determines which state the trunk is in and starts scanning for a change to the other state.

When a signal is recognized, the exec identifier at TVA.MPAAR+0 of the terminal process block (or TVA.MPBAR if terminal process B is chosen) is posted. A report code identifying the signal is then stored in the least significant nibble of TVA.MPAAR+1 (or TVA.MPABR) and the detection process continues.

SATVR applies only to PM loads. This opcode is valid for ATM and ADCM loads.

The format for SATVR is as follows:

SATVR !! control onhk_mask offhk_mask scan_mask

Where:

control is one of the following:

01 normal scan points

00 extension scan points

onhk_mask is the scan point state when on-hook

offhk_mask is the scan point state when off-hook

scan_mask is a mask showing which scan points to view.

For a DCM, the format for SATVR is as follows:

SATVR !! cont1 cont2 onhk_val offhk_val sc_id

Where:

cont1 is control byte 1, where:

- **bit 0** is the terminal process bit. When set to 0, terminal process A is active. When set to 1, terminal process B is active.
- **bit 1** is the midbit. When set to 0, the process will start normally. When set to 1, the process will start at a confirmed on-hook or off-hook state.
- **bit 2** is the fast off-hook option bit. When set to 0, normal mode is established. When set to 1, fast off-hook mode is established.
- **bits 3-7** are not used.

cont2 is control byte 2. This byte is not used.

onhk_val is the on-hook value of the scan point:

- **bit 0** bit on-hook value A
- **bit 1** bit on-hook value B
- **bits 2-7** value of 0.

Note: Either bit 0 or bit 1 is used at any one time and is specified by the scan mask.

offhk_val is the off-hook value of scan point. The value of bits 0-7 do not matter.

sc_id is the scan point identifier:

- **bit 0** 1 = scan ‘A’ bit
- **bit 1** 1 = scan ‘B’ bit

bits 2-7 are not used.

Note: Either bit 0 or bit 1 has to be set but not both.

For a TM, the format for SATVR is as follows:

SATVR !! cont1 cont2 onhk_val offhk_val sc_id

Where:

cont1 is the control byte 1. The bit assignments are as follows:

bit 0 is not used

bit 1 is the midbit. When set to 0, a normal start occurs. When set to 1, process starts at the state specified in bits 4-7.

bit 2 is the fast off-hook option bit. When set to 0, normal mode is established. When set to 1, fast off-hook mode is established.

bit 3 is the trunk on/off-hook bit. When set to 0, the trunk is assumed to be on-hook. When set to 1, the trunk is assumed to be off-hook.

Note: Bit 3 is only considered if the midbit (bit 1) is set.

bits 4-7 specify the machine start state. If the midbit (bit 1) is set, the starting state of the signal detection state machine is specified. Valid states are 0 to 8.

cont2 is control byte 2. The bit assignments are as follows:

bit 0 is the scan byte select. When set to 0, the extension scan byte is used. When set to 1, the normal scan byte is used.

bit 1 is not used

bit 2 is the SF trunk indicator. When set to 1, an SF trunk is used.

bits 3-7 are not used.

onhk_val is the on-hook value of scan point.

offhk_val is the off-hook value of scan point.

sc_id is the scan point identifier.

Note 1: Once the terminal process is started it will continue to report signals as they are received. The process will not terminate unless idled by another terminal process or primitive.

Note 2: In the fast off-hook mode, off-hooks are reported if the off-hook duration is greater than 150 ms.

Note 3: The detection mode can be switched between normal and fast off-hook while the terminal process is running.

Note 4: An error is generated if terminal 0 is used or if both A and B bits are enabled for scanning. (DCM only)

Note 5: An error is generated if SATVR is always run on terminal process A. (TM only)

47 ANDPARMS

ANDPARMS logically ANDs two values on the parameter stack and returns the result to the stack.

The format for ANDPARMS is as follows:

ANDPARMS !! n1 n2 ! n3

Where:

n1 is a one byte AND operator.

n2 is a one byte AND operator.

n3 is the result of a logical AND of the two operators.

48 ORPARMS

ORPARMS logically ORs two values on the parameter stack and returns the result to the stack.

The format for ORPARMS is as follows:

ORPARMS !! n1 n2 ! n3

Where:

n1 is a one byte OR operator.

n2 is a one byte OR operator.

n3 is the result of a logical OR of the two operators.

49 XORPARMS

XORPARMS logically EXCLUSIVE ORs two values on the stack and returns the result to the stack.

The format for XORPARMS is as follows:

XORPARMS !! n1 n2 ! n3
Where:

n1 is a one byte XOR operator.

n2 is a one byte XOR operator.

n3 is the result of a logical XOR of the two operators.

4A STKN0

STKN0 stacks the lower nibble from the terminal variable at offset bytes in the terminal process block.

An error is generated if the parameter stack overflows or if the offset is out of range.

The format for STKN0 is as follows:

STKN0 offset !! ! nibble
Where:

offset is the number of bytes offset from the top of the TVA. The range is #00 to #24.

nibble is the nibble to be stacked.

4B STKN1

STKN1 stacks the upper nibble from the terminal variable at offset bytes in the terminal process block.

An error is generated if the parameter stack overflows or if the offset is out of range.

The format for STKN1 is as follows:

STKN1 offset !! ! nibble
Where:

offset is the number of bytes offset from the top of the TVA. The range is #00 to #24.

nibble is the nibble to be stacked.

4C POPN0

POPN0 pops one byte from the parameter stack and puts the lower nibble into the lower nibble of the terminal variable at offset bytes in the terminal process block.

The format for POPN0 is as follows:

POPN0 offset !! data

Where:

offset is the number of bytes offset from the top of the TVA. The range is #00 to #24.

data bits 0-3 is the data nibble.

Bits 4-7 are not used.

4D POPN1

POPN1 pops one byte from the parameter stack and puts the lower nibble into the upper nibble of the terminal variable at offset bytes in the terminal process block.

The format for POPN1 is as follows:

POPN1 offset !! data

Where:

offset is the number of bytes offset from the top of the TVA. The range is #00 to #24.

data bits 0-3 is the data nibble.

Bits 4-7 are not used.

4E ADDIGS

ADDIGS adds n digits to the terminal process block digit area. Digits are added sequentially from the first available digit register indicated by the terminal digit count.

An error is generated if an invalid terminal is used or if the parameter stack overflows.

The digit count is incremented for every digit added. If the digit count reaches 15, the next available digit register is the first digit register. Any additional digits will overwrite the previous digit register contents.

The format for ADDIGS is as follows:

ADDIGS !! n digitn ... digit2 digit1

Where:

n specifies the number of digits to be added.

digitn...digit1 are the digits to be added.

4F TIMEDIF

TIMEDIF subtracts two 3-byte numbers to produce the call time.

The format for TIMEDIF is as follows:

TIMEDIF !! t1h t1m t1l t2h t2m t2l ! t3h t3m t3l
Where:

t1 is the minuend:

Note: In subtraction, the minuend is the number or quantity from which another number or quantity is subtracted.

t1h higher order byte

t1m middle order byte

t1l low order byte

t2 is the subtrahend:

Note: In a subtraction, subtrahend is the number or quantity subtracted from the minuend.

t2h higher order byte

t2m middle order byte

t2l low order byte

t3 is the result:

t3h higher order byte

t3m middle order byte

t3l low order byte

50 GETSC

GETSC stacks the terminal scan data onto the parameter stack.

The format for GETSC is as follows:

GETSC !! ! data
Where:

data is the terminal scan data.

For a DCM, the data returned is in the following form:

bit 0 is the A bit scan value

bit 1 is the B bit scan value

bit 2-7 equals 0.

51 GETXSC

GETXSC stacks the terminal extension scan data on the parameter stack.

The format for GETXSC is as follows:

GETXSC !! ! data

Where:

data is the terminal extension scan data.

For a DCM, all data equals 0.

52 SETSD

SETSD sets the terminal signal distributor point specified by the mask to the values specified by the final state. SETSD also operates the hardware and control points primarily for alarms.

The format for SETSD on a TM and a DCM is as follows:

SETSD !! mask final_state

Where:

mask identifies the signal distributor point. For a DCM:

- **bit 0** when set to 1, sets the A bit to final_state.
- **bit 1** when set to 1, sets the B bit to final_state.
- **bits 2-7** is not used.

final_state is the intended value of the signal distributor point. For a DCM:

- **bit 0** is the intended A bit value.
- **bit 1** is the intended B bit value.
- **bits 2-7** is not used.

The format for SETSDG on a LM and a RLM is as follows:

SETSDG !! mask sd_setting sd_group

Where:

mask identifies the signal distributor point.

sd_setting is the signal distributor point setting.

sd_group is the signal distributor point group.

53 SETXSD

SETXSD sets the terminal extension signal distributor point specified by the mask to the values specified by the final state.

The format for SETXSD is as follows:

SETXSD !! mask final_state

Where:

mask identifies the signal distributor point.

final_state is the intended value of the signal distributor point.

54 INCM

INCM includes n bytes of memory data, starting at the given address, in the currently open report. The data in the report is preceded by the parameters n, msbyad, and lsbyad.

An error is generated if a report is open or if the report exceeds the maximum length.

The format for INCM is as follows:

INCM !! n msbyad lsbyad

Where:

n is the number of bytes of memory data included in the report. For DCMs, TMs, and LMs, the maximum is 50 bytes. For RLMs, the maximum is 31 bytes.

msbyad is the most significant byte of memory data starting address.

lsbyad is the least significant byte of memory data starting address.

55 STKRX

STKRX copies n bytes from the terminal reflex buffer, starting at the specified offsets. The bytes are copied onto the parameter stack.

An error is generated if an invalid terminal number is specified (for DCM and TM, terminal 0 is invalid, for LM PID terminals 0 and 16 are invalid), if an offset and the number of bytes is greater than the reflex data area size (14 bytes), or if the parameter stack overflows.

STKRX can access 14 of the 16 reflex buffer bytes. The first two bytes (the message type) are not available.

The format for STKRX is as follows:

STKRX !! offset n ! datan...data2 data1

Where:

offset is the reflex buffer data area byte offset, from #00 to #0D.

n is the number of bytes to copy, from #01 to #0E. If 0 bytes are specified, data is not copied and an error is not generated.

56 SCSMON

SCSMON assigns a terminal process to monitor the given set of scan points for a change of state, which is the difference between the specified scan point and the expected current value specified in the SCSMON parameters. When the difference is constant for the duration of the filter time, SCSMON stores the exec identifier at TVA.MPAAR+0 and posts the exec. The new scan value is stored at TVA.MPAAR+0, replacing the exec identifier. The terminal process is then terminated.

SCSMON provides logic to support one-at-a-time origination control. If the terminal state indicator equals #FD, the scan change filter is greater than 50 ms, and the origination control flag equals 0, an origination exec is not posted and the terminal process is not terminated. The filter timer is restarted as if the scan change had just occurred and the terminal process continues.

The format for SCSMON is as follows:

SCSMON !! fltr scsv scsm mp

Where:

fltr is the filter time in units of 10 ms. This parameter specifies the minimum duration for a valid scan change. If the filter value is 0, no filtering is occurring.

scsv is the scan byte representing the expected current value. A deviation from this value is considered a change in state.

Note: For a DCM, bit 0 is the A bit scan value, bit 1 is the B bit value, and the value of bits 2-7 does not matter.

scsm is the scan byte mask. For a DCM:
bit 0 if set to 1, monitors the A bit for a change in the corresponding SCSV bit.
bit 1 if set to 1, monitors the B bit for a change in the corresponding SCSV bit.
bits 2-7 are not used

mp is the terminal process identifier. The bit values are:

bit 0 0 = terminal process A
1 = terminal process B
bits 2-7 are not used.

57 EXSMON

EXSMON assigns a terminal process to monitor the given set of extension scan points for a change of state. For more information, refer to 56 SCSMON on page 7-41.

The format for EXSMON is as follows:

EXSMON !! fltr scsv scsm mp

Where:

fltr is the filter time in units of 10 ms. This parameter specifies the minimum duration for a valid scan change. If the filter value is 0, no filtering is occurring.

scsv is a scan byte representing the expected current value. A deviation from this value is considered a change in state.

Note: For a DCM, bit 0 is the A bit scan value, bit 1 is the B bit value, and the value of bits 2-7 does not matter.

scsm is the scan byte mask. For a DCM:
bit 0 if set to 1, monitors the A bit for a change in the corresponding SCSV bit.
bit 1 if set to 1, monitors the B bit for a change in the corresponding SCSV bit.
bits 2-7 are not used

mp is the terminal process identifier. The bit values are:
bit 0 0 = terminal process A
1 = terminal process B
bits 2-7 are not used.

58 CSMMON

CSMMON monitors the channel supervision byte, similar to SCSMON.

CSMMON assigns a terminal process to monitor the specified channel supervision byte (SCSM) for a change in state, which is the difference between the specified bits of the SCSM and the value specified in the CSMMON parameters (SCSV). When the difference is constant for the duration of the filter time, CSMMON posts the exec identifier stored at TVA.MPAAR+0 (or TVA.MPBAR+0 if the terminal process B is used). The new scan value is then stored at TVA.MPAAR+0 (or TVA.MPBAR+0), replacing the exec identifier. The terminal process is then terminated.

The format for CSMMON is as follows:

CSMMON !! fltr scsv scsm mp

Where:

fltr is the filter time in units of 10 ms. This parameter specifies the minimum duration for a valid scan change. If the filter value is 0, no filtering is occurring.

scsv is a scan byte representing the expected current value. A deviation from this value is considered a change in state.

scsm is the scan byte mask, which defines the channel data byte bits to be monitored for change.

mp is the terminal process identifier. The bit values are:
bit 0 when set to 0, specifies terminal process A.
when set to 1, specifies terminal process B.
bits 2-7 are not used.

59 PATHCON

PATHCON sets up the channel PCM path according to the given function. Functions vary according to PM type. PATHCON also enables/disables various maintenance functions.

The format for PATHCON is as follows:

PATHCON !! function

Where:

function is one byte used to select a particular function.

5A SMTATS

SMTATS invokes the TM testing and traffic simulation terminal process on terminal process B.

SMTATS handles TATS signaling detection. The time between transition of the specified scan point is measured and stored in a table. The first transition is a change from the state the specified scan point had when SMTATS was started. This initial value is stored at PB.MPAAR+2 (TVA+2) and is reported at the CC.

SMTATS runs on terminal process A, killing terminal process B because SMTATS uses memory dedicated to terminal process B. Also, the background timer may be required if supervision timing is necessary.

An exec is posted if the expected number of transitions is recorded or when the supervision timer times out.

The format for SMTATS is as follows:

SMTATS counter_rate tmoval sc_id table_length

Where:

counter_rate is the base time for each table entry, in 5 ms units. The maximum time counter rate is 5 ms. A counter rate of 0 is treated as 1.

tmoval is the time-out time for the signal detection process, specified in units of 160 ms. A value of 0 indicates that no time-out time is specified for the terminal process. Therefore, SMTATS will run until the specified number of transitions is recorded.

sc_id is the scan point to monitor for a signal, from 0 to 7.

table_length is the threshold value for the number of entries (scan transitions) to be recorded before reporting to the CC. The maximum value is 16.

The terminal process block layout is as follows:

```
PB.MPAPB +0              = CURRENT COUNTER
         +1              = COUNTER_RATE
         +2              = TMOVAL
PB.MPAAR+0 (TVA+0)       = SMTATS END EXECID
         +1 (TVA+1)      = UNUSED FOR DCA
         +2 (TVA+2)      = INITIAL SC BIT VALUE
         +3 (TVA+3)      = SC_MASK
         +4 (TVA+4)      = NUMBER OF TRANSITIONS RECORDED
PB.MPBAR+0 (TVA+5)       = TABLE LENGTH
         +1 (TVA+6)      = TABLE ENTRY 1
         +2 (TVA+7)      = TABLE ENTRY 2
          .                .          .
          .                .          .
         *OVERWRITE DIGIT AREA*
          .                .          .
          .                .          .
PB.MISCB  +0 (TVA+20)    = TABLE ENTRY 16
```

5B NXTBIT

NXTBIT stacks the value of the bit using the values of the pattern in the PP large area.

All parameters are predefined in the TVA. The pattern length has a range of #00 to #0F, and the pattern has a maximum length of #10 bits.

An error is generated for stack overflow.

The following data is needed in the TVA:

- TVA.DIGRE + 4 = pattern length
- TVA.DIGRE + 5 = pattern (lsby)
- TVA.DIGRE + 6 = pattern (msby)
- TVA.DIGRE + 7 = current bit

The value of current bit is placed on the parameter stack. For example, if:

- pattern length = #05
- pattern (lsby) = #25
- pattern (msby) = not used
- current bit = #04

Then, processing NXTBIT three times will result in the following:

- NXTBIT !! ! #00
- (current bit = #05)
- NXTBIT !! ! #20
- (current bit = #00)
- NXTBIT !! ! #01
- (current bit = #01).

The format for NXTBIT is as follows:

NXTBIT !! ! data

5C S2DPREC

S2DPREC enables terminal process A for dial pulse (DP) reception.

This continuous terminal process only terminates if a last digit time-out, an abandon time-out, or an invalid digit occurs.

Digit collection is based on the timing of a wait-for-digit interval. The result of this interval is either a digit is received, or a digit is not received and the interval times out. If digit is received, the digit is stored, the digit count is incremented, and the next digit is anticipated. If a time-out occurs, either the digit_exec is posted and digits are anticipated, or the terminal process is idled and the last_digit_exec is posted.

DP digit reception process uses parameters that are defined each time the process is started. Some parameters are defined at initial setup, and others can be altered throughout the duration of the process.

The format for S2DPREC is as follows:

S2DPREC scmask polarity minwd

Where:

scmask is the scan point identifier, which is the set of scan points to be used in digit reception.

polarity is the polarity of the scan points indicating an on-hook.

minwd is the minimum acceptable pulse width in 5 ms units.

During outpulsing the terminal variable area is setup as follows:

```
MPAAR  +  0 =  exec_0    (digit_exec_id)
MPAAR  +  1 =  exec_1    (last_digit_exec_id)
MPAAR  +  2 =  exec_2    (abandon_exec_id)
MPAAR  +  3 =  stimeout (in 160 ms units)
MPAAR  +  4 =  ltimeout  (in 160 ms units)
DIGRE   +  0 =  digit1, digit0
DIGRE   +  1 =  digit3, digit2
DIGRE   +  2 =  digit5, digit4
DIGRE   +  3 =  digit7, digit6
DIGRE   +  4 =  digit9, digit8
DIGRE   +  5 =  digit11, digit10
DIGRE   +  6 =  digit13, digit12
DIGRE   +  7 =  digit_count, digit14
DIGRE   +  8 =  not used, # reported
DIGRE   +  9 =  stiming, ltiming
```

Following are brief explanations of the values given in the previous example.

- exec 0: generates a digit report and manipulates the value of STIMING and LTIMING, if necessary
- exec 1: is posted if (1) a wait for digit time-out has occurred, (2) the digit count has reached #F, or (3) an invalid digit is collected. The terminal process is idled before this exec is posted.
- exec 2: is posted if an abandon timeout has occurred. The terminal process is idled before this exec is posted.
- STIMEOUT: is the maximum amount of time to wait for digits when DIGIT_COUNT is greater than LTIMING but less than STIMING.
- LTIMEOUT: This is the maximum amount of time to wait for digits when DIGIT_COUNT is less than LTIMING.
- STIMING and LTIMING: LTIMING and STIMING are used together with DIGIT_COUNT to specify the wait-for-digit timing value and time-out type. In this way a user can have control over digit reporting and interdigit timing during digit collection. The DIGIT_COUNT threshold for posting the digit_exec is controlled by STIMING, and LTIMING dictates which interdigit interval is used (STIMEOUT or LTIMEOUT). Usually the value of LTIMING is less than that of STIMING

Notes

At the time of setup, the terminal process will initialize the terminal variable parameters as follows:

```
digit_count = 0
# reported = 0
ltiming     = #F
ltimeout    = #FF
```

The previous values become the default operating conditions of the reception process.

STIMEOUT and LTIMEOUT are given in units of 160 ms. A time-out value of 1 should be avoided since it specifies a 0 to 160 ms time-out.

If the values of LTIMING and STIMING are changed, the SIGNAL primitive should be called. This will cause the current wait-for-digit interval to restart using the new timing values.

5D S2DPOTP

S2DPOTP enables terminal process A for DP outpulsing.

During DP outpulsing, the terminal is in one of the following states:

- waiting for digit
- outpulsing
- interdigit timing
- stop dial timing.

Following the enabling of outpulsing, the terminal enters the wait-for-digit state. The terminal will remain in this state until it receives a NXTDIG, which initiates outpulsing. The terminal waits 10 to 20 ms before outpulsing the next digit. After the digit is outpulsed, the terminal waits 120 to 130 ms and posts an after_digit_exec. The SIGNAL command in the after_digit_exec indicates the exec has been processed and initiates interdigit timing.

Once interdigit timing is complete, the terminal process begins monitoring the specified scan point (with a 70 to 80 ms filter time). If the scan point goes to an on-hook value, an id_timeout_exec is posted and the terminal process returns to the wait-for-digit state. If the scan point goes to an off-hook value, a stop-dial or an answer has occurred and an off-hook exec is posted.

The SIGNAL command in the off-hook_exec advances the terminal process to the stop-dial-timing state, where the scan point will be monitored for an on-hook and a stop dial timer is started. If the scan point does not go on-hook before the timer expires, the terminal process is terminated and a stop_dial_timeout exec is posted. If an on-hook does occur, a go_dial_exec is posted and the terminal process returns to the waiting-for-digit state. This cycle is repeated until all digits have been outpulsed.

The format for S2DPREC is as follows:

S2DPOTP scmask sdmask polarity !! idtime sgtime

Where:

scmask is the scan point mask, which specifies the scan point to monitor for stop dial/answer off-hook.

sdmask is the signal distributor point mask which specifies the signal distributor point to use on outpulsing.

polarity specifies the scan point on-hook value. The bit values are:
bit 0 on-hook value
bits 1-7 are not used.

idtime specifies the interdigit timing value in units of 10 ms.

sgtime specifies the stop_dial timing value in units of 160 ms.

During outpulsing, the terminal variable area is setup as follows:

```
MPAAR + 0 = after_digit_exec_id/go_dial_exec_id
MPAAR + 1 = id_timeout_exec_id/
            stop_dial_time_exec_id
MPAAR + 2 = offhook_exec_id
DIGRE + 0 = digit1, digit0
DIGRE + 1 = digit3, digit2
DIGRE + 2 = digit5, digit4
DIGRE + 3 = digit7, digit6
DIGRE + 4 = digit9, digit8
DIGRE + 5 = digit11, digit10
DIGRE + 6 = digit13, digit12
DIGRE + 7 = digit_count, digit14
DIGRE + 8 = not used, next_digit_pointer
```

Following are brief explanations of the values given in the previous example:

- AFTER_DIGIT_EXEC is posted when the terminal process has successfully outpulsed a digit. The SIGNAL primitive must be included in the exec in order that outpulsing may continue to the interdigit timing state.
- ID_TIMEOUT_EXEC is posted when interdigit timing is complete and a stop_dial has not been received. The SIGNAL primitive must be included in the exec to advance outpulsing to the next wait-for-digit state.
- OFFHOOK_EXEC is posted whenever an off-hook is detected on the specified scan point. The SIGNAL primitive must be included in the exec in order to advance to the stop-dial-timing state.

- GO_DIAL_EXEC is posted if the scan point returns to on-hook during stop dial timing. The SIGNAL primitive must be included in the exec to advance outpulsing to the wait-for-digit state.
- STOP_DIAL_TIMEOUT_EXEC is posted if the stop dial timer expires. This exec will initiate stop dial/answer action. The terminal process is idled before the exec is posted.

5E SMFDTR

SMFDTR starts digit reception from a digital DTMF receiver or an analog MF receiver. Digits received are passed back to the originating peripheral processor by way of the Channel Supervision Message (CSM) if the CSF option is specified.

For the CSF option, the PP variable area MPAAR+1 should contain the EOT_EXIT exec and MPAAR+2 should contains the RCVR_ERROR exec.

The format for SMFDTR is as follows:

SMFDTR!! final_digit_mask csf_frst_dg_mask

Where:

final_digit_mask is the final digit mask, which controls the digits that are reported to the CC. The final digit mask will have no effect if the CSF option is used.

csf_frst_dg_mask is the first digit mask, which controls the digits that are sent to the CC. Bit 7 of CSF_FRST_DG_MASK is the CSF option flag. The first digit mask must always be specified.

For Digitone line handling, the CSF_FRST_DG_MASK should be set to #FF.

Notes

PP_MINOR_A_VAR expects the following values at offsets 0 to 4:

- digit exec identifier
- last digit exec identifier
- RCVR ERROR exec identifier
- short time-out and long time-out.

Digit collection is not started until a valid first digit is received. The digit exec is posted if the digit count has reached the value in short timing. The last digit exec is posted if a valid last digit is received. The RCVR ERROR exec is posted is the receiver detects reception trouble.

PP_LARGE_AREA is used to store the 15 digits plus digit counts and short and long timing values.

The digit count and number of digits reported are initialized to 0. The long timing and long time-out values are initialized to their maximum values.

5E UTRMFREC

UTRMFREC starts digit reception using a UTR DTMF receiver.

The first and last digit mask must always be specified.

UTRMFREC runs under the control of terminal process B and controls the start dial signaling to the far end office. The start dial signaling to the far end office trunk is performed when the assigned UTR channel is ready to receive digits. The delay dial on-hook signal and the wink dial off-hook signal are delayed until the digit reception process starts to avoid missing any digits.

The format for UTRMFREC is as follows:

UTRMFREC !! format_byte final_digit_mask first_dg_mask

Where:

format_byte is the format field. If bit 0 is set to true, start dial is driven by the exec found at TVA PP_LARGE_AREA+1 when the SP informs the MP that the UTR channel is ready to collect digits from the far end office.

If bit 1 is set to true, the signal set is DTMF. If bit 1 is not true, the signal set is MF.

If bit 2 is set to true, a request for a UTR channel will not be queued.

final_digit_mask defines the digits that qualify as final digits. If the most significant bit is set, this parameter indicates that BELL signaling is in effect. This parameter is used for ANI/ONI spill digit collection.

first_dg_mask defines the digits that qualify as first digits. If the most significant bit is set, this parameter indicates that wink signaling is in effect. This parameter is used for collecting the second of two strings of digits for an ANI/ONI spill. If this parameter is not set, reversal signaling is in effect.

Note: PP_MINOR_B_VARS should contain the digit_exec, PP_MINOR_B_VARS+1 should contain the last_digit_exec, and PP_MINOR_B_VARS+2 should contain the rcvr_error_exec.

5F S2MFOTP

S2MFOTP enables terminal process A for MF outpulsing.

During MF outpulsing, the terminal is in one of the following states:

- wait for digit
- outpulsing
- interdigit timing.

Immediately following the enabling of digit outpulsing, the terminal enters the wait-for-digit state, where it will remain until the NXTDIG command is received. NXTDIG causes the terminal to enter the outpulsing state, where the MF tone associated with the next digit is turned on. The tone remains on for 100 ms before an after_digit_exec is posted. The SIGNAL command contained in the after_digit_exec indicates the exec has been processed and interdigit timing should be initiated for the terminal process.

When interdigit timing is complete, and id_timeout_exec is posted. The SIGNAL command contained in the id_timeout_exec indicates that the exec has been processed. SIGNAL prompts the terminal process to reenter the wait-for-digit state. For all successive wait-for-digit states, off-hook scanning is enabled for stop dial/answer detection.

The cycle is repeated when the NXTDIG command is received; however, the tone duration is equal to the interdigit timing value.

If an off-hook occurs during the wait-for-digit state, the terminal process is terminated and an off-hook_exec is posted.

The format for S2MFOTP is as follows:

S2MFOTP scmask polarity !! idtime

Where:

scmask is the scan point mask, specifying which scan point to monitor for stop dial/answer off-hook.

polarity specifies the on-hook value of the scan point.

idtime specifies the interdigit timing value in units of 10 ms.

During outpulsing, the terminal variable area is set up as follows:

MPAAR + 0 = exec_0 (after_digit_exec_id)
MPAAR + 1 = exec_1 (id_timeout_exec)
MPAAR + 2 = exec_2 (off_hook_exec)
DIGRE + 0 = digit1, digit0
DIGRE + 1 = digit3, digit2
DIGRE + 2 = digit5, digit4
DIGRE + 3 = digit7, digit6
DIGRE + 4 = digit9, digit8
DIGRE + 5 = digit11, digit10
DIGRE + 6 = digit13, digit12
DIGRE + 7 = digit_count, digit14
DIGRE + 8 = not used, NEXT_DIGIT_POINTER

Following are brief explanations of the values given in the previous example.

- exec_0 is posted when the terminal process has successfully outpulsed a digit tone. The SIGNAL primitive must be included in the exec in order that outpulsing may continue to the interdigit timing state.
- exec_1 is posted when the terminal process has completed timing of the interdigit state. The SIGNAL exec must be included in the exec to prompt the terminal process to enter the wait-for-digit state.
- exec_2 is posted if a scan point off-hook is successfully filtered for 30-49 milliseconds during a wait-for-digit state. The exec will take stop dial/answer action. The terminal process is idled before this exec is posted.
- NEXT_DIGIT_POINTER is a pointer updated by the terminal process indicating which digit register (digit0 to digit14) contains the next digit for outpulsing.

60 GETBT

GETBT stacks the terminal board type.

For the TM, GETBT reviews the associated global location and determines which of the 2X45AA or AB cards are in the TM. The trunk identifier is put onto the parameter stack.

For the DCM, GETBT returns a value of 0 to the parameter stack.

An error is generated for stack overflow or for using terminal 0.

GETBT should not be used if digital trunks are connected to the TM. If GETBT is used and the card type is 2X45AA, the trunk identifier appears on the extension scan byte of all thirty terminals.

If GETBT is used on a TM and the card type is 2X45AB, the trunk timer and scan-matcher will be idled for the referenced terminal. Therefore, this primitive should not be used on an active trunk.

The format for GETBT is as follows:

GETBT !! ! data

61 CSMSET

CSMSET sets the channel supervision byte field, defined by the mask, to the given value.

An error is generated if an invalid terminal is used.

The format for CSMSET is as follows:

CSMSET !! mask value

Where:

mask is the channel data byte mask specifying which bit(s) of the Call Data Block to set to the specified value.

value is the value of Call Data Block bits specified in the mask.

62 TRAEX

TRAEX starts an exec trace for a given terminal and initializes the trace buffer to 0. The exec identifiers are stored in one of two specified trace buffers, each of which is capable of storing 32 exec identifiers. The buffers wrap around when full.

Exec traces may run on two terminal simultaneously if each terminal uses a different trace buffer.

The format for TRAEX is as follows:

TRAEX !! trml_no buffer_no

Where:

trml_no is the terminal identifier, specifying the terminal on which the exec trace is started. The value #FF turns the trace off.

buffer_no selects the buffer. A value of 00 specifies buffer 0 and 01 specifies buffer 1.

63 TRARP

TRARP includes the contents of an exec trace buffer in an open report. The 34 byte buffer contains:

- the terminal number associated with the terminal whose trace data is contained in the buffer.
- a pointer to the location in the buffer where the last exec identifier was written.
- 32 bytes of the exec identifier trace data.

An error is generated if no report is open or if the length of the buffer exceeds the available space in a report.

This primitive does not terminate an exec identifier trace.

The format for TRARP is as follows:

TRARP !! buffer_no

Where:

buffer_no specifies the buffer to be included in the report. A value of 00 specifies buffer 0 and 01 specifies buffer 1.

64 STAIDSCAN

Note: This primitive is not valid for DCMs.

STAIDSCAN starts an idle line scan for an off-hook on the currently referenced terminal.

The format for STAIDSCAN is as follows:

STAIDSCAN

65 STPIDSCAN

Note: This primitive is not valid for DCMs.

STPIDSCAN stops an idle line scan on the currently referenced terminal.

The format for STPIDSCAN is as follows:

STPIDSCAN

66 EXTOPGRP

EXTOPGRP extends itself to allow another section of opcodes to be available.

The format for EXTOPGRP is as follows:

EXTOPGRP valid opcode name opcodeparameters
Where:

valid opcode name is an opcode name that is valid for RLMs, DCMs, and XPMs.

opcodeparameters are parameters required by the opcode.

67 SST320

Note: This primitive is not valid for TMs and DCMs.

SST320 sets the state table for 320 lines.

The format for SST320 is as follows:

SST320 action_byte data0 data1...data39
Where:

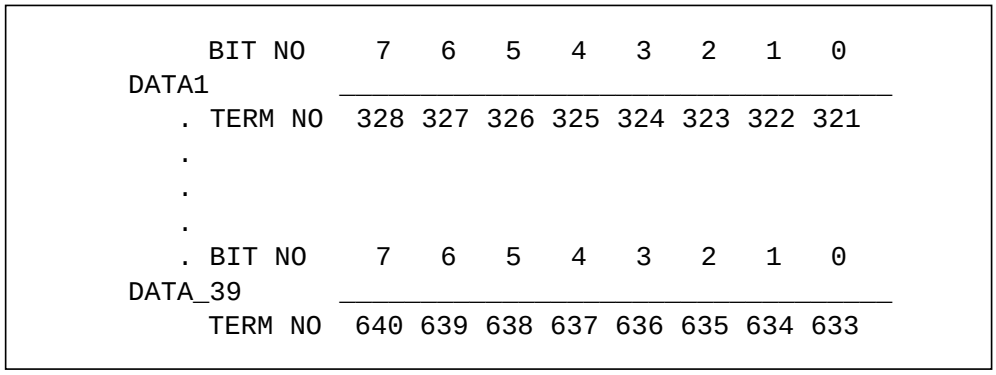
action_byte sets up the state table. The bit assignments are as follows:

bit 0 if set to 0, sets the state of lines 1 - 320 as shown in Figure 7-1.
if set to 1, sets the state of lines 321 - 640 as shown in Figure 7-2.

Figure 7-1xxx
Bit assignments when bit 0 is set to zero

	BIT NO	7	6	5	4	3	2	1	0
DATA0									
.	TERM NO	8	7	6	5	4	3	2	1
.									
.									
.									
.	BIT NO	7	6	5	4	3	2	1	0
DATA39									
	TERM NO	320	319	318	317	316	315	314	313

Figure 7-2xxx
Bit assignments when bit 0 is set to one



68 SARSUP

Note: This primitive is not valid for TMs and DCMs.

SARSUP starts the audible ringing process on a terminal.

The line loss setting is sent to the line card. If bit 6 of the control byte is set, PCM is connected to the line card (audible ringing provided from the trunk). If bit 6 is not set, terminal process A is started to perform 2 seconds on, 4 seconds off audible ring. In either case, the terminal is supervised.

SARSUP requires that the TVA be set up with MPAAR+0 equal to the ORIGINATOR_ABANDON_EXEC and MPAAR+1 equal to the TERMINATOR_ANSWER_EXEC.

The format for SARSUP is as follows:

SARSUP !! line_char control_byte

Where:

line_char data used for setting the gain before audible ringing is applied. Bits 1 to 3 are the loss settings to be sent to the line card.

control_byte is bit 6 which, when set to 1, allows PCM to be connected.

69 SRINGSUP

This primitive is NOT valid for TMs and DCMs.

Note: This primitive is not vailid for TMs and DCMs.

SRINGSUP starts terminal process A to perform ringing setup and supervision.

The format for SRINGSUP is as follows:

SRINGSUP !! line_char ring_char rev_ring_char

Where:

rev_ring_char specifies the revertive ring pattern. The bit assignment is as follows:
BIT 7 when set to 1, immediately initiates revertive ringing.
BIT 6-3 is the revertive ringing pattern code. If the revertive ringing pattern is 0FH, no revertive ringing occurs.
BIT 2-0 is the revertive ringing type.

ring_char specifies the ringing characteristics. The bit assignments are as follows:
BIT 7 when set to 1, provides immediate ringing.
BIT 6-3 ring pattern code.
BIT 2-0 ringing type.

line_char specifies the line characteristics. The bit assignments are as follows:
BIT 7 tip and ring. A value of 1 indicates tip.
BIT 6 the balanced network number to be sent to the line circuit.
BIT 5 is used by SCMs for revertive ringing. This bit is set to true if the originator is on the same side as the terminator.
BIT 4 is not used.
BIT 3-1 is the loss setting to be sent to the line circuit.
BIT 0 is not used.

SRINGSUP expects the following setup in the TVA:

MPAAR + 0 = RINGING_TIMEOUT_VALUE (1.28 sec units, 0 = ignore)
MPAAR + 1 = RINGING_TIMEOUT_EXEC
MPAAR + 2 = RING_BLOCKED_EXEC
MPAAR + 3 = ORIGINATOR_HANG_UP_EXEC
MPAAR + 4 = RING_TRIP_EXEC

SRINGSUP expects the following setup in the Protected Global Area:

RINGING_FAULT_HANDLER_EXEC (RGFLTEX(OF))
RINGING_PARAMETER_TABLES

6A WRTP

Note: This primitive is not vailid for TMs and DCMs.

WRTP writes one byte into the checksum protected ringing generator global data area.

The format for WRTP is as follows:

WRTP sec_key parameter_id offset !! data
Where:

sec_key security key, which is equal to #ED.

parameter_id identifies the data area that will be written to. A value of 00 specifies the ringing generator data area. A value of 01 specifies the signaling processor data area.

offset specifies the byte offset from the start of the ringing generator global data area to the area where one byte of data will be written. The range is from #00 to #BF.

data one byte of data.

6B ORGSUSUP

Note: This primitive is not vailid for TMs and DCMs.

ORGSUSUP starts a terminal process to perform origination set up and supervision.

ORGSUSUP contains five phases. Two phases are set up data for the next phase of the call and the remaining three perform functions similar to S2DPREC, LFNTG, XMNTG, SARSUP, SRINGSUP, and SACTSUP.

The first phase of ORGSUSUP is the synchronization process. The sync process establishes integrity or takes the appropriate action if integrity is not found.

The next phase is the OCPS0 Process. OCPS0 is a transitional process between the sync process and the audible ringing process. When synchronization is established, OCPS0 is invoked and will rearrange the terminal process block area to suit the requirements of the audible ringing process.

The audible ringing process supplies audible ringing to the originator. Balance network and gain relays are updated, and the originator is put into the active scan list (the signaling processor scans for on-hook).

The next phase is the OCPS1 process. OCPS1 is a transitional process between the audible ringing process and the active supervision process. When the terminator answers, OCPS1 is invoked. The terminal process

block area is then rearranged to suit the requirements of the active supervision process, and PCM is connected.

Finally, the active supervision process is performed on calls in the active talking state. The active supervision process monitors for on-hook, flash or wink (if flash/wink option is set), and takes appropriate action. The same code is used for both the originator and the terminator; however, different actions can be taken based on the control byte originator/terminator bit.

The format for ORGSUSUP is as follows:

ORGSUSUP !! line_char exint_msby exint_lsby control_byte

Where:

line_char specifies the line characteristics. The bit assignments are as follows:

BIT 7 is not used.

BIT 6 is the balanced network number to be sent to the line circuit.

BIT 5 is used by SCMs for revertive ringing. This bit is set to true if the originator is on the same side as the terminator.

BIT 4 is not used.

BIT 3-1 is the loss setting to be sent to the line circuit.

exint_msby is the most significant byte of the integrity value transmitted and expected after synchronization is achieved.

exint_lsby is the least significant byte of the integrity value transmitted and expected after synchronization is achieved.

control_byte is the control byte. The bit assignments are as follows:

bit 7 when set to 1, initiates the flash option during audible ringing.

bit 6 when set to 0, supplies audible ringing. When set to 1, suppresses audible ringing.

bit 0 when set to 1, initiates the flash option during active supervision.

6C TERSUSUP

Note: This primitive is not vailid for TMs and DCMs.

TERSUSUP starts the terminal process to perform termination set up and supervision.

TERSUSUP contains five phases. Two phases are set-up data for the next phase of the call and the remaining three perform functions similar to other primitives.

The first phase is the synchronization process. The sync process used in the terminator setup and supervise primitive is the same one as used in ORGSUSUP. The sync process can differentiate whether the caller was an originator or terminator by the phase code.

The next phase is the TCPS0 process. TCPS0 is a transitional process between the sync process and the ringing terminal process. TCPS0 rearranges the terminal process block area to suit the requirements of the ringing terminal process.

The ringing terminal process is a complicated process, dependent on the line card type and analog and digital tip and ring.

The TCPS1 process is a transitional process between the ringing terminal process and the active supervision process. When the terminator has answered (the originator is still off-hook), TCPS1 will rearrange the terminal process block area to suit the requirements of the active supervision process and will transmit off-hook to the originator.

Finally, the active supervision process used in the terminator setup and supervise primitive is the same one used in ORGSUSUP. The active supervision process can determine whether the caller was an originator or terminator by the phase code.

The format for TERSUSUP is as follows:

TERSUSUP !! line_char ring_char rev_ring_char txint_msby txint_lsby control_byte

Where:

line_char specifies the line characteristics. The bit assignments are as follows:
BIT 7 tip and ring. A value of 1 indicates tip.
BIT 6 the balanced network number to be sent to the line circuit.
BIT 5 is used by SCMs for revertive ringing. This bit is set to true if the originator is on the same side as the terminator.
BIT 4 is not used.
BIT 3-1 is the loss setting to be sent to the line circuit.

ring_char specifies the ringing characteristics. The bit assignments are as follows:
BIT 7 when set to 1, provides immediate ringing.
BIT 6-3 ring pattern code.
BIT 2-0 ringing type.

rev_ring_char specifies the revertive ring pattern. The bit assignment is as follows:
BIT 7 when set to 1, initiates revertive ringing.
BIT 6-3 is the revertive side ringing pattern code.
BIT 2-0 is the revertive ringing type number.

txint_msby is the most significant byte of the integrity value transmitted and expected after synchronization is achieved.

txint_lsby is the least significant byte of the integrity value transmitted and expected after synchronization is achieved.

control_byte is the control byte. When bit 0 is set to 0, no flash option is available. When set to 1, the flash option is available.

6D SACTSUP

Note: This primitive is not valid for TMs and DCMs.

SACTSUP initiates talking supervision by the terminal process.

If bit 2 of the control byte is reset, CDB interrupt is disabled; otherwise, CDB mismatch interrupt is enabled for on-hook or off-hook, depending on the value of bit 4 of the control byte.

The signaling processor active scan process is started for on-hook or off-hook scanning, depending on the setting of bit 3 of the control byte.

An error is generated if the terminal process is started on invalid channels 0, 16, 32, and so on.

The format for SACTSUP is as follows:

SACTSUP !! control_byte
Where:

control_byte defines the active supervision attributes. The bit assignments are as follows:

bit 0 when set to 0, no flash option is enabled. When set to 1, the flash option is enabled.

bit 1 when set to 0, supervision takes place on the originating side. When set to 1, supervision takes place on the terminating side.

bit 2 when set to 0, single-sided supervision is initiated. When set to 1, double-sided supervision is initiated.

bit 3 when set to 0, active scanning for on-hook is started. When set to 1, active scanning for off-hook is started.

bit 4 when set to 0, if bit 2 is set, monitors for supervision on-hook. When set to 1, if bit 2 set, monitors for supervision off-hook.

bit 5 when set to 0, no wink option is enabled. When set to 1, the wink option is enabled.

bits 6-7 are not used.

During active supervision the terminal variable area is setup as follows:

```
MPAAR + 0 = onhook_exec_id
MPAAR + 1 = exit_offhook_exec_id
MPAAR + 2 = opposite_end_hangup_exec_id
MPAAR + 3 = flash_action_exec_id
MPAAR + 4 = offhook_exec_id
```

Following are descriptions of the terms used in the previous example:

- ONHOOK_EXEC: is posted when the active scan process determines that the line has gone on-hook. If exit off-hook timing is initiated, the terminal process is idled. Otherwise, active scanning is started for off-hook detection.
- EXIT_OFFHOOK_EXEC: is posted when exit off-hook timing has expired. This exec is used only for double-sided supervision. After the exec is posted, active scanning is stopped and the terminal process is idled.
- OPPOSITE_END_ONHOOK_EXEC: is posted when a supervision on-hook is detected from the far end. This exec is used only for double-sided originating supervision. Prior to posting the exec, exit off-hook timing is started.
- FLASH_ACTION_EXEC: is posted when a flash or a wink is detected.
- OFFHOOK_EXEC: is posted when active scanning detects an off-hook on the line. After posting the exec, active scanning for on-hook is started.

6E KSTIME

Note: This primitive is not valid for TMs and DCMs.

KSTIME stops the 10 ms short background timer on the referenced channel and removes any queued time-out messages.

An error is generated if an invalid channel is referenced, such as channel 0, 16, 32, and so on.

The format for KSTIME is as follows:

KSTIME

6F KLTIME

Note: This primitive is not valid for TMs and DCMs.

KLTIME stops the 10 ms long background timer on the referenced channel and removes any queued time-out message.

An error is generated if an invalid channel is referenced, such as channel 0, 16, 32, and so on.

The format for KLTIME is as follows:

KLTIME

71 STANITEST

Note: This primitive is not valid for TMs and DCMs.

STANITEST starts the ANI/coin terminal process in ANI mode. Once the test is complete, the results are written to TVA.MISCB+0 and the exec at TVA.MPAAR+2 is posted.

An error is generated if the control-byte value is not 00 or 01.

The format for STANITEST is as follows:

STANITEST !! control-byte

Where:

control-byte specifies either 2-party or 4-party ANI test. A value of 00 indicates a 2-party test and 01 indicates a 4-party test.

During ANI testing, the terminal variable area must be set up as follows:

MPAAR + 0 = ANI time-out value in 2.56 sec units
MPAAR + 1 = ANI time-out exec-id
MPAAR + 2 = ANI report exec-id

The results of the ANI test that are written to TVA.MISCB+0 have the following bit assignments:

- bit 0-1 is the test result. For a 2-party test, a value of 00 indicates the ring party and a value of 01 indicates the tip party. For a 4-party test, a value of 00 indicates the ring party 1, a value of 01 indicates the tip party 4, a value of 10 indicates ring party 3, and a value of 11 indicates tip party 2.
- bit 7 when set to 1, indicates the test was successful.

72 STCOINF

Note: This primitive is not valid for TMs and DCMs.

STCOINF processes coin phone functions and starts the ANI/coin terminal process.

The format for STCOINF is as follows:

STCOINF !! function number

STCOINF expects the following values in the TVA:

MPAAR + 0 = COIN FUNCT TIMEOUT TIME IN 2.56 SECS
MPAAR + 1 = COIN FUNCT TIMEOUT EXEC ID
MPAAR + 2 = COIN FUNCT REPORT EXEC ID

73 STTABLE

Note: This primitive is not valid for TMs and DCMs.

STTABLE manipulates the state table.

The format for STTABLE is as follows:

STTABLE !! data_byte action_byte ! data_read

Where:

data_byte indicates one of the following actions:

If action 0 or action 3 are specified as the action_byte value, the state table will be copied to the occupancy table of this unit if data_byte is set to 0, of the mate peripheral if the data_byte is set to 1, or of both peripherals if the data_byte is set to 2.

If action 1 is specified, call processing is active when bit 0 of the data_byte value is set to 1. Call processing is inactive when bit 0 of the data_byte value is set to 0.

If action 2 is specified, the value of the data_byte does not matter.

action_byte performs on of the following actions:

0 reset either the mate, this unit, or both sections of the state table to call processing inactive and start the LM audit process.

1 write the state of the referenced terminal.

2 read the state of the referenced terminal and put the result on the stack.

3 copy either the mate, this unit, or both sections of the state table to the occupancy table.

STTABLE expects the following setup in the protected global area:

TLAUDEX (2E) = TERMINAL_AUDIT_EXEC

74 ACTVCON

Note: This primitive is not valid for TMs and DCMs.

ACTVCON controls LM activity.

The format for STTABLE is as follows:

ACTVCON security_key !! control_byte

Where:

security_key is the security key, which is equal to #ED.

control_byte specifies one of the following actions:

C1 ACTIVITY_ON, which starts the sanity process on its own controller.

C2 SERVICE_MATE, which requests signaling processor to extend the idle scan cycle over the mate lines. SERVICE_MATE also informs the ringing generator process to start servicing the mate generator.

C3 ACTIVITY_OFF, which stops the sanity process for its own controller. An activity can be held for up to 200 ms.

C4 STOP_SERVICING_MATE, which requests the signaling processor and the ringing generator process to stop servicing the mate and kill the processes for the mate lines.

C7 INITIALIZE_LM, which initializes the connection memory and the idle scan table.

C8 FORGET_MATES, which stops the idle scan on inactive or idle mate lines. FORGET_MATES also stops all terminal processes associated with any active mate lines.

C9 ACTIVITY_ONLY, which assumes activity on its own bay. An idle scan is not started.

CA CLEAN_UP_LM, which cleans up connection memory and channels associated with mate lines.

CC FORGT_OWN, which stops the idle scan on inactive or idle lines. FORGT_OWN also stops all terminal processes.

CD ACT_MATE, which assumes activity of mate lines. An idle scan is not started.

C5 KILL_ALL_TERM_PROCESSES (local option)

C6 KILL_MATE_TERM_PROCESSES (local option)

Notes

To prevent a surge of call termination and abandon messages to the CC, the ACTIVITY_OFF option causes all active terminals to be reset. The primitive examines all channel process blocks for an associate terminal not equal to 0. If one is found,an internal sequence is posted for that channel. The internal sequence resets the terminal and invokes the ACVCON primitive again with the KILL_ALL_TERM option. The KILL_ALL_TERM option examines all subsequent channel process blocks and posts the internal sequence until all channel process blocks have been examined.

Similar activity is performed when STOP_SERVICING_MATE is processed. However, only terminals related to the mate are reset.

75 DOSA

Note: This primitive is not valid for TMs and DCMs.

DOSA performs service analysis.

The format for STTABLE is as follows:

DOSA !! function

Where:

function when 80H is specified, dial tone speed measurement is started. When set to 00H, the dial tone speed measurement is cancelled.

76 ATCHN

Note: This primitive is not valid for TMs and DCMs.

ATCHN reassociates a specified channel to the referenced terminal.

An error is generated if an invalid terminal is referenced, an invalid channel is referenced, or if the channel to be reassociated to a terminal is the same channel associated previously.

The format for ATCHN is as follows:

ATCHN !! chnl_no

Where:

chnl_no is a valid channel number.

77 SETXTBL

Note: This primitive is not valid for TMs and DCMs.

SETXTBL sends the pretranslation tables to the peripheral.

The format for SETXTBL is as follows:

SETXTBL size data0...datan

Where:

size is the size of the message.

data0...datan is the set of data in the translation table.

78 STMA

Note: This primitive is not valid for TMs and DCMs.

STMA initiates a maintenance action.

STMA copies the data from parameters solic_no to path-5 into the memory block and sends the data as a message to the maintenance master. The maintenance master locates a free maintenance process block. If the maintenance master succeeds, a maintenance process is started for the respective test.

For continuous test requests, the LM_TEST_STATUS exec (id = E1) is invoked to indicate to the CC whether the test was started successfully. If a continuous test fails, the LM_CONT_TEST_BAD exec (id = E2) is invoked to report the test result to the CC.

For single shot test requests, the LM_TEST_STATUS exec (id = E1) is only invoked when the test cannot be performed because no maintenance process blocks exist. At termination of a single shot test the LM_SINGLE_TEST exec (id=E0) is invoked to send the test result to the CC.

The format for STMA is as follows:

STMA BYTE_COUNT, SOLIC_NO, MAINT_ACT_NO, CONTROL_BYTE
PATH-0, PATH-1, PATH-2, PATH-3, PATH-4, PATH-5 .

Where:

byte_count specifies the number of bytes of the message

solic_no is the CC solicitor id. The information is stored in maintenance process block to be used for subsequent maintenance report.

maint_act_no is the id of the maintenance action to be started

control_byte indicates whether the maintenance action is a single shot processing or a continuous process.

path-0 specifies the source (ringing generator #, tone number, and so on)

path-1 specifies the channel

path-2 specifies the network digroup

path-3 specifies the internal digroup

path-4 specifies own or mate bay

path-5 specifies the line circuit number.

The message format is as follows:

0	**LINK-LOW**	
1	**LINK-HI**	
2	**ACTION CODE (=STMA)**	
3	**PID-LOW**	
4	**PID-HI**	
5	**D0**	**BYTE_COUNT**
6	**D1**	**SOLIC_NO**
7	**D2**	**MAINT_ACTION_ID**
8	**D3**	**CONTROL_BYTE**
9	**D4**	**PATH-0**
10	**D5**	**PATH-1**
11	**D6**	**PATH-2**
12	**D7**	**PATH-3**
13	**D8**	**PATH-4**
14	**D9**	**PATH-5**

79 CANMA

Note: This primitive is not valid for TMs and DCMs.

CANMA cancels a maintenance action.

CANMA compiles and sends a message to the maintenance master. If the maintenance process block occupancy table indicates that a maint_act_id is processing in a maintenance process block, the server message queue is purged of all messages coming from that maintenance process.

The format for CANMA is as follows:

CANMA maint_act_id

7A WROT

Note: This primitive is not valid for TMs and DCMs.

WROT writes to the occupancy table.

The format for WROT is as follows:

WROT item_class item_no_lsby item_no_msby !! write_code

Where:

item_class is one of the following:
30 own_bis
31 mate_bis

item_no_lsby is the item number least significant byte.

item_no_msby is the item number most significant byte.

write_code is one of the following:
00 item will be deleted from the occupancy table.
FF item will be entered into the occupancy table.

7B RDOT

Note: This primitive is ot valid for TMs and DCMs.

RDOT reads the occupancy table.

The format for RDOT is as follows:

RDOT item_class item_no lsby item_no_msby !! ! presence_indic
Where:

item_class is one of the following:
30 own_bis
31 mate_bis

item_no_lsby is the item number's least significant byte.

item_no_msby is the item number's most significant byte.

presence_indicator is one of the following:
00 item is not present
FF item is present.

7C FREEMPB

Note: This primitive is ot valid for TMs and DCMs.

FREEMPB frees the associated maintenance process block if the referenced process is stopped.

The format for FREEMPB is as follows:

FREEMPB

7D RRTP

Note: This primitive is ot valid for TMs and DCMs.

RRTP reads the ringing and system parameters.

The format for RRTP is as follows:

RRTP parameter identity offset
Where:

parameter identity is the identity of the parameter to be read.

offset is the offset from the base of the protected ringing generator parameter area (RGPARM).

7E STKMVA

Note: This primitive is ot valid for TMs and DCMs.

STKMVA stacks n bytes from the associated maintenance process block, starting with the location at offset from the beginning of the process data area.

The format for STKMVA is as follows:

STKMVA offset !! nbytes

7F STKMAST

Note: This primitive is ot valid for TMs and DCMs.

STKMAST stacks three bytes that are contained in a maintenance message from the maintenance master.

The format for STKMAST is as follows:

STKMAST !! ! maint_act_status maint_action_id solic_no

Where:

maint_act_status is the maintenance action status to be stacked.

maint_action_id is the maintenance action identifier.

solic_no is the solicitation number.

Following is the message format:

	0	LINK-LSBY
	1	LINK-MSBY
	2	ACTION CODE (EXEC POSTING)
	3	PID-LSBY
	4	PID-MSBY
D0	5	EXEC ID
D1	6	SOLIC_NO
D2	7	MAINT_ACTION_ID
D3	8	MAINT_ACTION_STATUS

7B PPSEGINFO

PPSEGINFO contains information on the next load segment in new peripherals during bootstrapping.

The format for PPSEGINFO is as follows:

PPSEGINFO

7C PPFUNCTION

PPFUNCTION loads the extension memories of the peripheral processors. This opcode is applicable only during bootstrapping.

The format for PPFUNCTION is as follows:

PPFUNCTION

7D PPSTATUS

PPSTATUS updates the peripheral processor status. This opcode is applicable only during bootstrapping.

The format for PPSTATUS is as follows:

PPSTATUS

7E PPLOAD

PPLOAD loads data into the peripheral processor. This opcode is applicable only during bootstrapping.

The format for PPLOAD is as follows:

PPLOAD

7F PPRUN

PPRUN starts program processing during bootstrapping.

The format for PPRUN is as follows:

PPRUN

PPTEST 79

Note: This primitive is applicable only to new peripherals.

PPTEST tests the peripheral processor and memory in new peripherals during bootstrapping.

The format for PPTEST is as follows:

PPTEST

80+(n-1) STKIn

STKIn copies n bytes of immediate data onto the parameter stack. STKIn has a range from 80 to 87.

This primitive fails on parameter stack overflow.

A maximum of eight bytes can be stacked at one time.

The format for STKIn is as follows:

STKIn im0 im1...imn !! ! imn...im1 im0

Where:

im0 im1...imn are the immediate data bytes.

88+(n-1) STKPn

STKPn is used in an exec to obtain parameters on the stack below the return address stored by an NVKXEC primitive. STKPn has a range from 88 to 8F.

Starting at the first parameter associated with STKPn, which is the offset above the base of the stack before the last NVKXEC, n parameters are put on the top of the stack. STKPn does not reorganize the order of the parameters that are already on the stack. The parameters being read are not disturbed.

This primitive fails on stack overflow.

A minimum of eight bytes can be stacked at a time.

The format for STKPn is as follows:

STKPn offset !! ! parmn ... parm1

Where:

offset specifies the byte offset above the base of the previous stack from where n parameters are put onto the current stack. The first byte above the stack base is at offset zero.

90+(n-1) STKVn

STKVn stacks n bytes from the terminal variable area of the terminal process block, starting with the byte at offset bytes from the start of the global area.

STKVn has a range from 90 to 9F.

This primitive fails on stack overflow.

A maximum of 16 bytes can be stacked at one time.

The format for STKVn is as follows:

STKVn offset !! ! varn ... var1
Where:

offset specifies the byte offset from the start of the terminal area from where n variables are put onto the parameter stack. (The first byte from the start of the terminal variable area is at offset 0.)

A0+(n-1) STKGn

STKGn stacks n bytes from the global variable area, starting with the byte at offset bytes from the start of the global area.

This primitive has a range from 90 to 9F.

This primitive fails on stack overflow.

A maximum of 16 bytes can be stacked at one time.

The format for STKVn is as follows:

STKVn offset !! ! globn...glob1
Where:

offset specifies the byte offset from the start of the global area from where n variables are put onto the parameter stack. (The first byte from the start of the global area is at offset 0.)

B0+(n-1) INCn

INCn POPs n parameters from the stack and copies them into a report in the order parm1, parm2... parmn.

INCn has a range of B0 to BF.

This primitive fails on stack underflow, if no report is open, or on report buffer overflow.

The format for INCn is as follows:

INCn !! parmn...parm1
Where:

parmn...parm1 are the parameter bytes for inclusion in a report. A maximum of 16 bytes can be included in a report at one time.

C0+(n-1) STKDn

STKDn stacks n bytes from the terminal data area or the reflex data area, starting with the byte at offset bytes from the start of the data area. The most significant nibble of the offset is used to select the source data area.

STKDn has a range from C0 to C7.

This primitive fails on stack overflow. Errors are also generated if the terminal data area index is out of range, the reflex data area index is out of range, or an attempt is made to access the reflex data when a reflex message is enabled on that terminal.

The format for STKDn is as follows:

STKDn offset !! ! datan...data1

Where:

offset selects the data area (either terminal data area or reflex data area) and specifies the byte offset within the data area. A value of 0N references the data byte at offset N of the terminal data area. The maximum value of N is the number of data bytes per terminal minus 1.

A value of #FN references the data byte at offset N of the reflex data area. The maximum value of N is F, which is the reflex buffer size.

Notes

The terminal data area and the reflex data are not treated as contiguous by the firmware.

A maximum of 8 bytes can be stacked at one time, unless the data area size limits the number of bytes that can be stacked.

C8+(n-1) POPDn

POPDn POPs n bytes from the stack into the terminal data area or the reflex data area, starting at offset bytes from the top of the data area. The most significant nibble of the offset is used to select a data area.

POPDn has a range of C8 to CF.

This primitive fails on stack underflow. Errors are also generated if the terminal data area index is out of range, the reflex data area index is out of range, or an attempt is made to POP data when a reflex message is enabled on that terminal.

The terminal data area and the reflex data are not treated as contiguous by the firmware.

The format for POPDn is as follows:

POPDn offset !! parmn...parm1

Where:

offset selects the data area (either terminal data area or reflex data area) and specifies the byte offset within the data area. A value of 0N references the data byte at offset N of the terminal data area. The maximum value of N is the number of data bytes per terminal minus 1.

A value of #FN references the data byte at offset N of the reflex data area. The maximum value of N is F, which is the reflex buffer size.

D0 TKIDLER

Note: This primitive is used for DTCs/LTCs only.

TKIDLER is used by flow and overload control to reset data structures. The trunk state is set to idle.

The format for TKIDLER is as follows:

TKIDLER control ! idle_preformed_flag

Where:

control when bit 0 is set, invokes the XPM code to idle the trunk. If the trunk is idled, a value of 1 will be returned on the stack and the TVA for that trunk will be overwritten.

If bit 0 is not set, no parameter is returned on the stack.

idle_preformed_flag indicates whether the trunk has been idled.

D1 CPINTENT

CPINTENT is used for high level call processing intent messages sent between the CC and the XPM.

CPINTENT should NOT be used for messaging to DTCs or any other small memory XPM because the primitive is not supported.

D2 ISDDPR

ISDDPR is used for the Incoming Start-to-Dial Delay (ISDD) unsolicited OM.

The ISDDPR primitive is implemented only for the terminal types of AB_TRK_TERM and AB_250_TERM, through the AB trunk primitive decoder table.

The format for ISDDPR is as follows:

ISDDPR function ! data

Where:

function is one of the following types:

0 - func_unld_data: Unload trunk data. A byte following the format byte contains the start/signal type information for the current trunk being processed by TPT. The data is stored in the trm_signal array based on the terminal number, and in sc_signal array based on the scan terminal number. This function is used primarily during the RTS supervision of a trunk supporting ISDD.

1 - func_save_time_stamp: Save the time stamp of a trunk seizure, currently in the scan report buffer, into a field in the active terminal header block (ATHB) of the trunk. The field, **isdd_time**, contains the least significant word of the MP's time stamp at the point when the AB scanner detected the seizure (the scanner includes the time stamp in its report of the seizure). This function is used from within the start execs that handle trunk originations.

2 - func_calc_isd_delay: Calculate the incoming start-to-dial delay based on the current MP time and the time stamp stored in the trunk ATHB. If the delta is greater than the ISDD_threshold value then a delay is pegged in the appropriate OM based on the trunk start/signal types. This function is performed in the execs that are used to send the trailing edge of the start-to-dial signals, and in the execs that handle abandons before completion of the start-to-dial signal.

data if the function is **func_unld_data format**, the signal/start types of the trunk is specified.

If the function is **func_calc_isd_delay**, the ISDD state the trunk will be set to is specified. This is used to determine if an abandon or the trailing edge of the start-to-dial signal is being handled by TPT. The capability for ISDDPR to make the proper state transition is provided.

D3

Not defined

D4

Not defined

D5

Not defined

D6 DMS250EXTOP

DMS250EXTOP is the extended opcode group for DMS250 signaling primitives. The signaling primitives are specified in the second opcode byte following the DMS250EXTOP primitive.

The format for DMS250EXTOP is as follows:

DMS250EXTOP extended_opcode opcode_parameters

Where:

extended_opcode is one of the following:

01 REORIG
02 STRMON
03 IDLESTR

opcode_parameters are any parameters required by the extended opcode.

D7 MAIDEXTOP

MAIDEXTOP is the extended opcode group byte of the Socotel signaling primitives. The Socotel primitives are indicated by a second opcode byte following this primitive.

The format for MAIDEXTOP is as follows:

MAIDEXTOP extended_opcode opcode_parameters

Where:

extended_opcode is one of the following:

01 SOCREC
02 MTRRCVR
03 SOCBCODE
04 SOCOTP
05 SVMAPPOL
07 MSPARE

opcode_parameters are any parameters required by the extended opcode.

D8 VTIME

VTIME starts a background timer that has a period of 1.28 seconds.

The format for VTIME is as follows:

VTIME !! exec_id t
Where:

exec_id is the exec that is invoked if the timer expires.

t is the number of 1.28 second periods that the background timer will run before expiring.

D9 SIGGEN

Note: This primitive is for DTC A/B trunks only.

SIGGEN is the signal generator primitive.

When a signal identifier is specified, SIGGEN generates a sequence of on- and off-hooks on the specified signal distributor point (s). The timer terminal process is used with a function similar to STIME. The specified exec_id is stored in the TVA at the offset PP_TIMER_EXEC. When the signal has been sent, the exec_id is posted. The timing value is used for flash and wink. The timing value is specified in units of 10 ms and will be used as the first or the only timing value within SIGGEN. If the value of the time is not set, default values are used.

Supported signals are as follows:

- PRE-EMPT TO SEIZE (military)
- PRE-EMPT TO CLEAR (military)
- FLASH, PULSE ONHOOK
- WINK, PULSE OFFHOOK.

The format for SIGGEN is as follows:

SIGGEN !! exec_id time sd_id sig_id
Where:

exec_id is an exec identifier

time is the timing value

sd_id is the signal distribution point on which a sequence of on- and off-hooks are generated

sd_id is the signal identifier

DA INTRASWITCH

Note: This primitive is for LTCs only.

INTRASWITCH sets up a connection within the peripheral (RLCM) between the two agents.

The format for INTRASWITCH is as follows:

INTRASWITCH trmnl_id ext_byte control_byte

Where:

trmnl_id is the terminal identifier of the other agent.

ext_byte is the extension byte of the TID (for P-phones).

control_byte is the control byte.

DC PATHCON2

PATHCON2 connects the PCM path in new peripherals.

The format for INTRASWITCH is as follows:

PATHCON2 !! function link data

Where:

function is one of the following (for DTCs and LTCs):

- **#10** enable DS-1 mtce
- **#20** monitor DS-1 card in/out status
- **#30** disable DS-1 mtce
- **#40** set DS-1 line to normal & enable full mtce
- **#50** loop DS-1 line
- **#70** loop DS-1 line & disable DS-1 mtce
- **#90** set outgoing alarm register (loop mode only)
- **#a0** loop DS1 toward remote end
- **#b0** reset outgoing alarm register (loop mode only)
- **#c0** set DS-1 status to unequipped
- **#d0** clock synchronization - sample & report
- **#e0** clock synchronization - sample only
- **#f0** clock synchronization - stop sampling

link is the DS-1 link.

data indicates synchronization, which specifies register 0 or 1 for sampling (hardwired registers).

DE SENDM

SENDM sends an intent message to a task in an LGC/DTC or LCM.

The format for SENDM is as follows:

SENDM dest_comid src_comid length send_byte1...send_byte

Where:

dest_comid is the destination Communication Identifier (COMID) of the task.

src_comid is the source COMID, which is usually the CC.

length is the number of data bytes that follow.

send_byte1...send_byten are the data bytes.

DF ASSCH2

Note: ASSCH2 is only used for new peripherals that support more than 256 network channels.

ASSCH2 associates the specified channel with the terminal.

ASSCH2 is similar to ASSCH (refer to 2D ASSCH on page 7-20), except that ASSCH2 is used for channel numbers in the range 256-511, whereas ASSCHN is used for channels 0-255.

The format for SENDM is as follows:

ASSCH2 !! (ch-no mod 256)

E0+(n-1) POPVn

POPVn POPs n bytes from the stack into the terminal variable area, starting at offset bytes from the start of the variable area.

POPVn has a range of E0 to EF.

This primitive fails on stack underflow.

The format for POPVn is as follows:

POPVn offset !! parmn...parm1

Where:

offset specifies the byte offset from the start of the terminal variable area to where n variables are popped from the parameter stack. (The first byte from the start of the terminal variable area is at offset 0.)

parmn...parm1 are parameter bytes to be popped into the terminal variable area. A maximum of 16 bytes can be accessed at a time.

F0+(n-1) POPGn

POPGn POPS n bytes from the stack into the global variable area, starting at offset bytes from the start of the variable area.

POPGn has a range from F0 to FF.

This primitive fails on stack underflow.

The format for POPGn is as follows:

POPGn offset !! parmn...parm1
Where:

offset specifies the byte offset from the start of the global variable area to where n variables are popped from the parameter stack. (The first byte from the start of the global variable area is at offset 0.)

parmn...parm1 are parameter bytes to be popped into the global variable area. A maximum of 16 bytes can be accessed at a time.

List of terms

ANI

Automatic Number Identification

ATD

Audio Tone Detector

Automatic Number Identification (ANI)

A system in which a calling number is identified automatically and transmitted to the AMA office equipment for billing.

CC

Central Control

Central Control (CC)

Comprises the data processing functions of the DMS-100 Family, with associated Data Store and Program Store.

Channel Supervision Message (CSM)

A message received and transmitted continuously on each connected voice channel of a Peripheral Module. The CSM contains a connection data byte which includes the Channel Supervision Bit and an integrity byte which issues call-path integrity.

CI

Command Interpreter

Command Interpreter (CI)

A support operating system component that functions as the main interface between machine and user. Its principal roles are:

1 to read lines entered by a terminal user

2 to break each line into recognizable units

3 to analyze the units

4 to recognize command input-numbers on the input lines

5 to invoke these commands.

CSM

Channel Supervision Message

Dial Pulse (DP)

Method of transmitting signaling information from a telephone set or trunk circuit. Dial pulses are generated by alternately opening and closing a Contrast with Dual-Tone Multifrequency Dialing.

DIOMSG

Display I/O Message. Formats and displays an I/O message.

Directory Number (DN)

The full complement of digits required to designate a subscriber's station within one NPA - usually a three-digit Central Office code followed by a four-digit station number.

DN

Directory Number

DP

Dial Pulse

DTMF

Dual Tone Multifrequency Signaling

Dual Tone Multifrequency Signaling (DTMF)

A signaling method employing set combinations of two specific voice-band frequencies, one of which is selected from a group of four low frequencies, and the other from a group of three or four relatively high frequencies.

Exec

A string of primitives that define PP procedures.

Extract

A user removes specific messages from the I/O message stream.

FPE

Feature Processing Environment

Hook

1) An element added to old software to enable it to recognize new features that did not exist when the older software was written. 2) An element of a software module that accesses other optional modules. Responsible for evaluating a conditional dependence.

IMD:

Intercepted Message Dispatcher Process

Intercepted Message Dispatcher Process (IMD)

Receives, logs, records, and dispatches intercepted I/O messages.

Insert

A user puts a message into the I/O message stream.

I/O

Incoming and/or outgoing.

Intercept

1) A user records what messages are sent in the I/O message stream. 2) A user prevents messages from reaching a PM or the CC.

LEN

Line Equipment Number

Line Equipment Number (LEN)

Composed of the site, frame number, unit number, drawer number, and circuit number. For example, the LEN HOST 00 0 05 08 has the site HOST, frame number 00, unit number 0, drawer number 05, and circuit number 08.

MF

Multifrequency

MIDC

Message Intercept and Dispatch Control

Message Intercept and Dispatch Control (MIDC)

Coordinates activities between the user and the IMD process.

Multifrequency (MF)

A method that makes use of pairs of standard tones to transmit signaling codes, digit pulsing, and coin-control signals.

Node

Any unit that can accept or originate messages.

Node Number

A system assigned number unique to a node.

Opcode

Identifies a specific operation to be performed.

Primitive

String of opcodes for specific operations that the PP will perform.

PCM

Pulse Code Modulation

Peripheral Module (PM)

A generic term referring to all hardware modules of the DMS-100 Family systems that provide interfaces with external line,trunk, or service facilities. PMs contain Peripheral Processors which perform local routines, thus relieving the load on the Central Processing Unit.

Peripheral Module Intercept System Test (PMIST)

A debugging tool that traces messages between the Peripheral Modules.

Peripheral Processor (PP)

Hardware devices contained in the Peripheral Modules that perform local processing functions independent of the Central Processing Unit. PPs are driven by Read-Only Memory in the PM, thus releasing CPU run-time for higher level activities.

PM

Peripheral Module

PMIST

Peripheral Module Intercept System Test

PMIST Up (PUPI)

An exec file containing CI commands that invoke PMIST.

PP

Peripheral Processor

Pulse Code Modulation (PCM)

Representation of an analog waveform by coding and quantizing periodic samples of the signal such that each element of information consists of a binary number representing the value of the sample.

PUPI

PMIST Up

Reflex Buffer

Contained in each terminal of a PM. Holds a two byte report type and 14 bytes of primitive instructions.

Reflex Flag

Signals CC instructions have been executed or an exec has been processed.

Report

An incoming message.

Scan Points

Read only bits in the trunk logic circuit which indicate the status of a feature of the hardware.

SD Points

Signal Distribution Points

Signal Distribution Points (SD Points)

Writeable bits in the Trunk Logic Circuit which usually correspond to relays in the hardware and are used to control the action of the hardware.

SFDEV

Store File Device

Terminal

1) Refers to both the interface circuit on a circuit board mounted in a PM unit and the device it is connected to. Devices include telephone sets, trunk circuits, and data links. 2) An external connection to the DMS system.

Terminal Identifier

The node number and the terminal number.

Terminal Number

A number given to a specific terminal attached to a node.

Terminal Process

Procedures usually set up to perform signalling and supervision tasks. The process performs its function until a predefined event, such as end of digits, occurs.

User

A person, organization, or other group that uses the services of a DMS switch.

DMS-100 Family

PMIST

Technical Assistance Manual

Publication number: TAM-1001-007
Product release: BCS36 and up
Document release: Standard 08.02
Date: December 1993

Printed in the United States of America