BSC 3270 PROGRAMMER'S MANUAL

September 1990
Version 2.0

PREFACE

This manual is intended to provide a programmer's guide to the BSC 3270 Monitor/Emulation programs. General programming information is provided in the Programmer's Reference Manual. Information contained in this manual is machine independent.

This manual is not intended to provide basic user instruction, but rather addresses the issues of writing test programs using the Interactive Test Language (ITL). Refer to the machine specific User Manual for a quick reference to the basic operation of the protocol tester.

P/N 6000-1205

IDACOM Electronics Ltd.
A division of Hewlett-Packard

4211 - 95 Street
Edmonton, Alberta
Canada T6E 5R6
Phone: (403) 462-4545
Fax: (403) 462-4869

TABLE OF CONTENTS

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

TABLE OF CONTENTS [continued]

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The BSC 3270 Monitor is implemented in accordance with the IBM 3270 Information Display System: 3274 Control Unit Description and Programmer's Guide, GA23–0061–2. publication for point–to–point (3780) and multipoint (3705/3274) protocol procedures.

It is not a state driven monitor, i.e. it does not have knowledge of expected events. Rather the monitor decodes and reports information on received messages and lead changes. Filters, RAM capture, and disk recording are also available.

The BSC Emulation is implemented as a state–driven protocol emulation together with an integral protocol monitor. In the multipoint environment, the COMM/3705 (communications controller) and CLST/3274 (cluster controller) emulations have been set up to run as:

- an automatic simulation which operates in accordance with IBM 3270 Information Display System: 3274 Control Unit Description and Programmer's Guide, GA23–0061–2. publication for point–to–point (3780) and multipoint (3705/3274) protocol procedures;
- a semi–automatic tester. The test manager is used to build and execute test scenarios for generation of errors, and to test responses to all protocol messages; and
- a manual tester. The test is controlled from the user's keyboard.

The COMM/3705 and CLST/3274 emulations can function in the following two BSC modes:

- Control mode
- Text mode

NOTE
Transparent monitor mode and transparent mode are not available in the BSC Monitor or Emulation program.

All user test scripts are written in the ITL language. Test programs are made up of sequences of ITL commands (or 'words') which exchange data and parameters via a Last In First Out (LIFO) stack. All commands consume zero or more parameters from the stack (input) and/or leave results on the stack (output). These commands have a stack effect comment shown beside the definition of the command to define its input and output parameters.

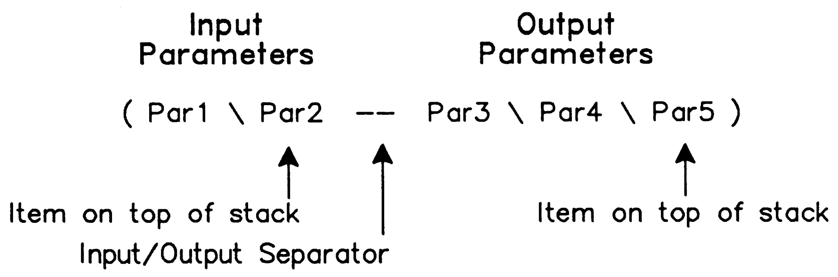


Figure 1–1 Sample Stack Comment

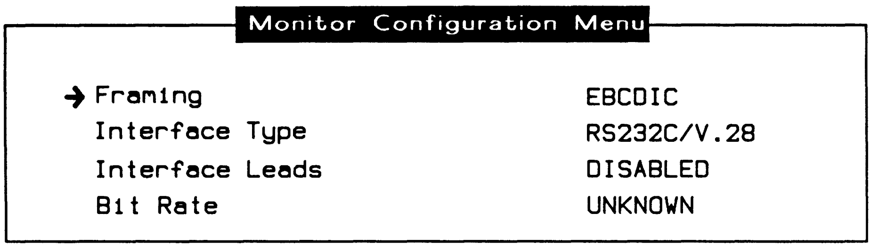
NOTE
See Appendix D for further explanation of stack parameters.

Sample complete test scripts are supplied in Section 13. These test scripts are also supplied on disk with the application program.

The BSC applications can be controlled remotely from a terminal. All commands described in this manual can be entered from the remote terminal's keyboard followed by a ↵ (RETURN). The application processes the remote command and returns the 'ROK' prompt to the remote terminal. The remote terminal must be connected to the modem port on the back of the tester. To configure the application for remote control, refer to the Programmer's Reference Manual.

2 MONITOR CONFIGURATION

This section describes the commands associated with each item on the Monitor Configuration Menu.



```
Monitor Configuration Menu

→ Framing                         EBCDIC
  Interface Type                  RS232C/V.28
  Interface Leads                 DISABLED
  Bit Rate                        UNKNOWN
```

Figure 2–1 Monitor Configuration Menu

RECONFIGURE (--)
Initialized the BSC protocol for the monitor and configures the physical interface.

NOTE
Use RECONFIGURE once after all physical changes are made.

→ *Framing*
BSC–EBCDI (--)
Uses EBCDIC framing (default) with the following characteristics:

- NRZ (standard) clocking
- 8 bits per character
- No parity
- EBCDIC character set
- Sync character of hex 32
- Interframe fill is mark
- Rest idle is mark
- DCD control is on
- CRC calculation according to CRC–16
- Strip sync is on

EBCDIC function key

BSC−ASCII (--)

Uses ASCII framing with the following characteristics:

- NRZ (standard) clocking
- 7 bits per character
- odd parity
- ASCII character set
- Sync character of hex 16
- Interframe fill is mark
- Rest idle is mark
- DCD control is on
- CRC calculation according to VRC/LRC
- Strip sync is on

ASCII function key

→ *Interface Type*

IF=V28 (--)

Selects the V.28/RS−232C connector (default) and electrically isolates the other connectors on the port.

RS232C/V.28 function key

IF=V11 (--)

Selects the V.11/X.21 connector and electrically isolates the other connectors on the port.

RS422/V.11 function key

IF=V35 (--)

Selects the V.35 connector and electrically isolates the other connectors on the port.

V.35 function key

IF=V36 (--)

Selects the V.36/RS−449 connector and electrically isolates the other connectors on the port.

RS449/V.36 function key

NOTE

A WAN tester has a V.28, V.11, and either a V.35 or V.36 connector. These commands are only applicable if the program is running on a WAN interface.

→ *Interface Leads*
Individual or all interface leads can be enabled or disabled. Leads must be enabled for test manager detection.

ENABLE_LEAD (lead identifier --)
Enables the specified lead. Refer to the Programmer's Reference Manual for a list of supported leads for each interface type.

Example:
Enable the request to send lead.
```
IRS ENABLE_LEAD
```

DISABLE_LEAD (lead identifier --)
Disables (default) the specified lead. Refer to the Programmer's Reference Manual for a list of supported leads for each interface type.

Example:
Disable the clear to send lead.
```
ICS DISABLE_LEAD
```

ALL_LEADS (-- lead identifier)
Enables/disables all leads supported on the currently selected WAN interface. ALL_LEADS must be used with ENABLE_LEAD or DISABLE_LEAD.

Example 1:
Enable all leads for the current interface.
```
ALL_LEADS ENABLE_LEAD
```

ENABLE function key

Example 2:
Disable all leads for the current interface.
```
ALL_LEADS DISABLE_LEAD
```

DISABLE function key

→ *Bit Rate*
The interface speed is measured, in bits per second, directly from the physical line.

INTERFACE−SPEED (--address)
Contains the current bit rate (default value is 9600) and is used by the monitor to calculate throughput measurements.

NOTE
It is not necessary to set the interface speed to successfully monitor data.

3 MONITOR ARCHITECTURE

The BSC Monitor program monitors live data, saves data to capture RAM or disk, and displays data in a number of different formats. Data can be passed through filters which limit the displayed, captured, or recorded data.

3.1 Live Data

The monitor application receives events from the interface or internal timer and processes them as shown in Figure 3–1.

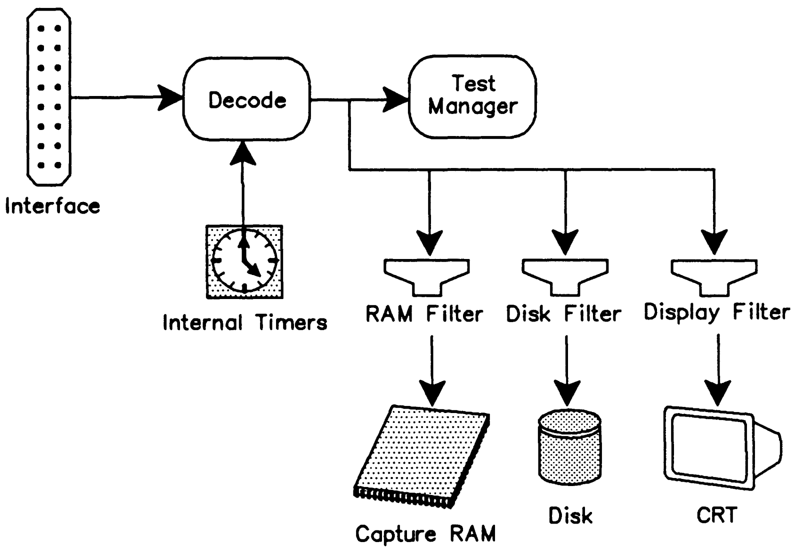


Figure 3–1 BSC Monitor Data Flow Diagram – Live Data

By default, the BSC Monitor captures data in the capture RAM buffer and displays it on the screen in a short format report.

Display topic
Live Data function key

MONITOR (--)
Selects the live data mode of operation. All incoming events are decoded and displayed in real–time.

3.2 Playback

Data (both protocol and lead information) can be examined in an offline mode using either capture RAM or disk as the data source.

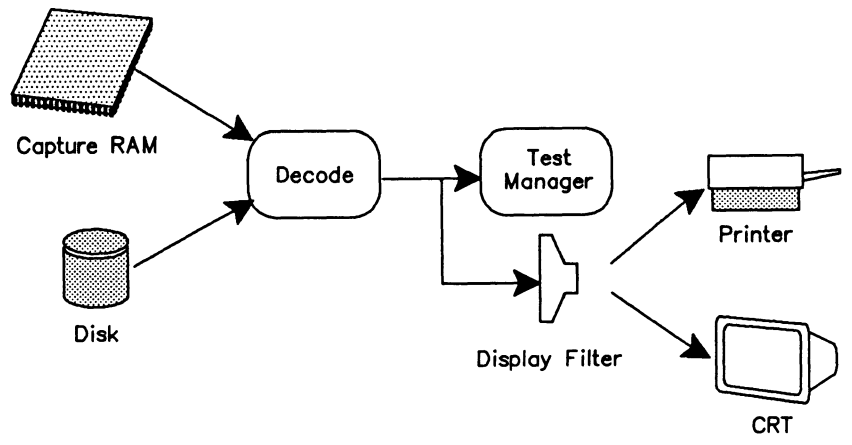


Figure 3–2 BSC Monitor Data Flow Diagram – Offline Processing

FROM_CAPT HALT
Display topic
Playback RAM function key

FROM_DISK HALT PLAYBACK
Display topic
Playback Disk function key

HALT (--)
Selects the playback mode of operation. Data is retrieved from capture RAM or disk, decoded, and displayed or printed. Capture to RAM is suspended in this mode.

FROM_CAPT (--)
Selects the capture buffer as the source for data transfer.

FROM_DISK (--)
Selects a disk file as the source for data transfer.

PLAYBACK (--)
Opens a data recording file for playback. When used in the Command Window, the filename can be specified as part of the command.

Example:
```
PLAYBACK DATA1
```

NOTE
When PLAYBACK is used in a test script, the filename must be specified with =TITLE.

=TITLE (filename--)
Specifies the name of the file to open for disk recording or disk playback.

Example:
Obtain playback data from disk.

```
FROM_DISK               ( Select disk file as data source )
HALT                    ( Enter halt mode for playback )
" DATA3" =TITLE         ( Select file to be played back )
PLAYBACK                ( Open file for playback )
```

Playback Control

The following commands control display scrolling.

FORWARD or F (--)
Scrolls one line forward on the screen.

⇓ (Down arrow)

BACKWARD or B (--)
Scrolls one line backward on the screen.

⇑ (Up arrow)

SCRN_FWD or FF (--)
Scrolls one page forward on the screen.

CTRL ⇓

SCRN_BACK or BB (--)
Scrolls one page backward on the screen.

CTRL ⇑

TOP (--)
Positions the display at the beginning of the playback source.

CTRL SHIFT ⇑

BOTTOM (--)
Positions the display at the end of the playback source.

CTRL SHIFT ⇓

3.3 Simultaneous Live Data and Playback

Live data can be recorded to disk while playing back data from capture RAM.

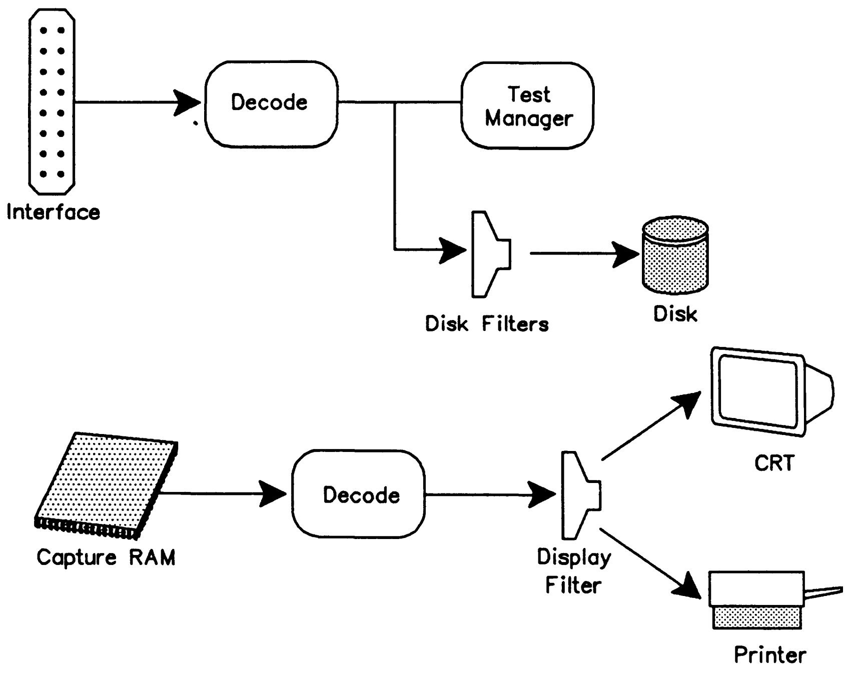


Figure 3–3 BSC Monitor Data Flow Diagram – Freeze Mode

FROM_CAPT FREEZE
Capture topic
Record to Disk function key
Display topic
Playback RAM function key

FREEZE (--)
Enables data to be recorded to disk while data from capture RAM is played back.

4 CAPTURE RAM

This section describes the data flow diagram for capture to RAM and lists the commands available for test scripts. Data stored in either capture RAM or disk can be played back as described in Section 3.2. Data stored in capture RAM can be transferred to disk.

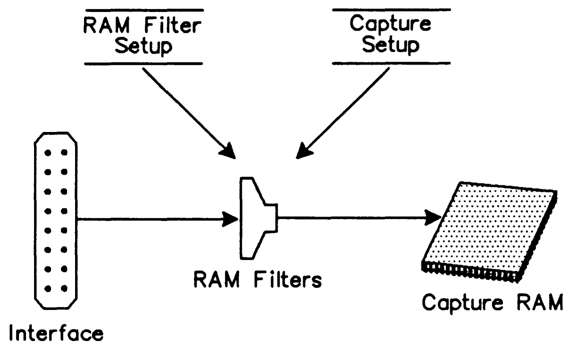


Figure 4–1 BSC Data Flow Diagram – Capture to RAM

4.1 Capturing to RAM

CAPT_ON (--)
Saves live data in capture RAM (default).

Capture topic
Capture to RAM function key (highlighted)

CAPT_OFF (--)
Live data is not saved in capture RAM.

Capture topic
Capture to RAM function key (not highlighted)

CAPT_WRAP (--)
Initializes capture RAM so that new data overwrites (default) old data after the capture buffer is full (endless loop recording).

Capture topic
Recording Menu
→ *When Buffer Full*
WRAP function key

CAPT_FULL (--)
Initializes capture RAM so that capturing stops when the buffer is full.

Capture topic
Recording Menu
→ *When Buffer Full*
STOP function key

WARNING
CAPT_FULL and CAPT_WRAP erase all data in capture RAM.

CLEAR_CAPT (--)
Erases all data currently in capture RAM.

Capture topic
Clear function key

4.2 Transferring from RAM

Data can be transferred from capture RAM to disk, and printed as it is played back. To transfer data to disk, a data recording must be opened using the RECORD and CTOD_ON commands prior to using TRANSFER. To transfer data from capture RAM to the printer, the PRINT_ON command must first be issued. The data being transferred is displayed on the screen.

TRANSFER (--)
Transfers data from the selected data source.

Capture topic
Save RAM to Disk function key

QUIT_TRA (--)
Abruptly terminates the transfer of data from capture RAM to disk.

Capture topic
Save RAM to Disk function key

TRA_ALL (--)
Transfers the entire contents of capture RAM (default) when the TRANSFER command is used.

Capture topic
Save RAM to Disk function key
All function key

TRA_START (--)

Selects the starting block for transfer and is used with TRA_END when a partial transfer is desired. Use the cursor keys to locate the desired starting block prior to calling TRA_START. TRA_START selects the last scrolled block as the initial starting block for transfer.

Capture topic
Save RAM to Disk function key
Set Start function key

TRA_END (--)

Selects the final block for transfer and is used with TRA_START when a partial transfer is desired. Use the cursor keys to locate the desired final block prior to calling TRA_END. TRA_END selects the last scrolled block as the final starting block for transfer.

Capture topic
Save RAM to Disk function key
Set End function key

SEE_TRA (--)

Displays the timestamps for the initial and final blocks selected for transfer in the Command and Test Script Windows.

Example:
Open a data file with the filename 'DATA1' and transfer all data from capture RAM to disk. After the transfer is complete, turn off data recording.

```
FROM_CAPT                         ( Designate capture RAM as data source )
HALT                              ( Enter playback mode )
" DATA1" =TITLE                   ( Assign filename DATA1 )
RECORD                            ( Open data recording )
CTOD_ON                           ( Enable capture transfer to disk )
TRA_ALL                           ( Transfer all data )
TRANSFER                          ( Transfer data from capture to disk )
DISK_OFF                          ( Turn off data recording )
```

To Disk

CTOD_ON (--)

Enables transfer of data from capture RAM to disk when data source is playback RAM and a data recording file is open.

CTOD_OFF (--)

Disables transfer of data from capture RAM to disk (default) when data source is playback RAM.

To Printer

PRINT_ON (--)

Prints data lines as displayed during playback from either capture RAM or disk. No printout is made when the source is live data. The printer must be configured on the Printer Port Setup Menu under the **Setup** topic on the Home processor.

Print topic
Print On function key

PRINT_OFF (--)

Data is not printed during playback (default).

Print topic
Print Off function key

Example:
Transfer all data from capture RAM to the printer.

```
FROM_CAPT                    ( Designate capture RAM as data source )
HALT                         ( Enter playback mode )
PRINT_ON                     ( Enable printing )
TRA_ALL                      ( Transfer all )
TRANSFER                     ( Transfer data to printer )
```

5 DISK RECORDING

Live data from the interface can be recorded to either a floppy or hard disk. Data stored in either capture RAM or disk can be played back as described in Section 3.2. Data stored in capture RAM can be transferred to disk as described in Section 4.2.

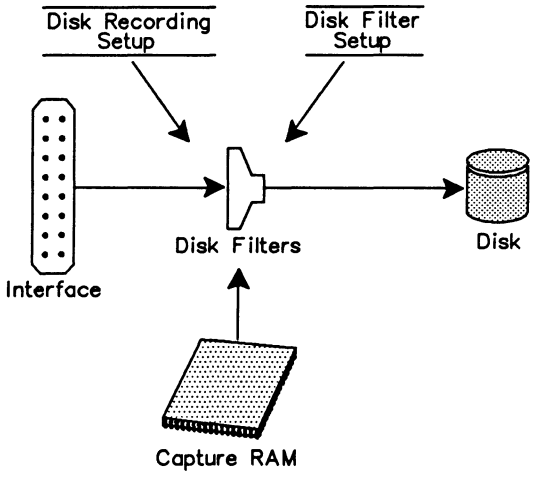


Figure 5–1 BSC Data Flow Diagram – Recording to Disk

DISK_WRAP (--)
Selects disk recording overwrite (default).

Capture topic
Recording Menu
→ *When File Full*
WRAP function key

DISK_FULL (--)
Turns off disk recording overwrite. Recording continues until the data recording file is full.

Capture topic
Recording Menu
→ *When File Full*
STOP function key

WARNING
DISK_WRAP and DISK_FULL must be called prior to opening a recording with the RECORD command. If called while recording is in process, the status of the disk recording overwrite for this recording session will not change.

RECORD (--)

Opens a data recording file. When used in the Command Window, the filename can be specified as part of the command.

Example 1:

```
RECORD DATA1
```

Capture topic
Record to Disk function key (highlighted)

NOTE
When RECORD is used in a test script, the filename must be specified with =TITLE. Because of the relatively long time required to open a disk file (especially on a floppy drive), RECORD should not be used within time critical portions of a test script.

Trace report lines are included in the data file when an application requests start and end recording. The information in these traces identifies the traffic type and application program used while the data was being recording.

Example:

```
Recording Start : BSC Emul COMM   WAN Port 1 RS232-C
V1.3-1.3  Rev 0                   PT500 - 24          SN# 01-261

Recording End   : Bsc Emul COMM   WAN Port 1 RS232-C
V1.3-1.3  Rev 0                   PT500 - 24          SN# 01-261
```

DISK_OFF (--)

Live data is not recorded to disk. The current disk recording is closed.

Capture topic
Record to Disk function key (not highlighted)

NOTE
Refer to the Programmer's Reference Manual for multi-processor disk recording.

DIS_REC (--)

Momentarily suspends data recording. The data recording file remains open but no data is saved to disk.

Capture topic
Record to Disk function key (highlighted)
Suspend Recording function key (highlighted)

ENB_REC (--)

Enables data recording. The data recording file remains open and live data is recorded to disk.

Capture topic
Record to Disk function key (highlighted)
Suspend Recording function key (not highlighted)

6 DISPLAY FORMAT

The BSC Monitor and Emulation applications can display data from the line (live data), from capture RAM, or a disk recording in the following display formats:

- Hexadecimal
- Character
- Short
- Complete
- Split
- Trace Statements

The data flow diagram for displaying and printing data, as well as commands available for test scripts, are described in this section.

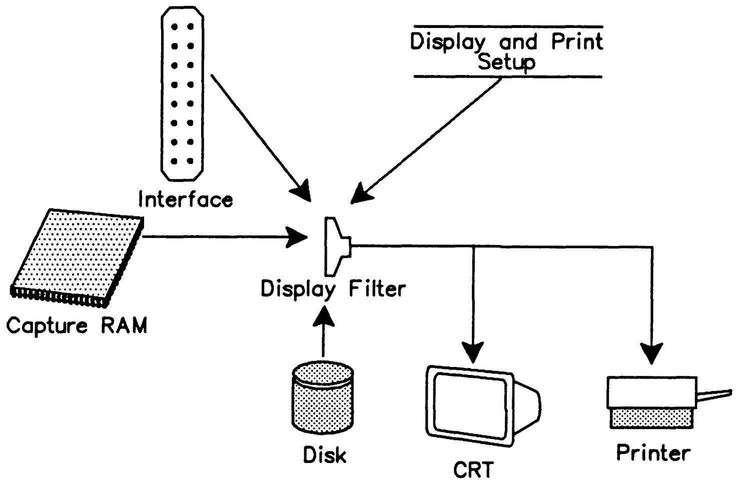
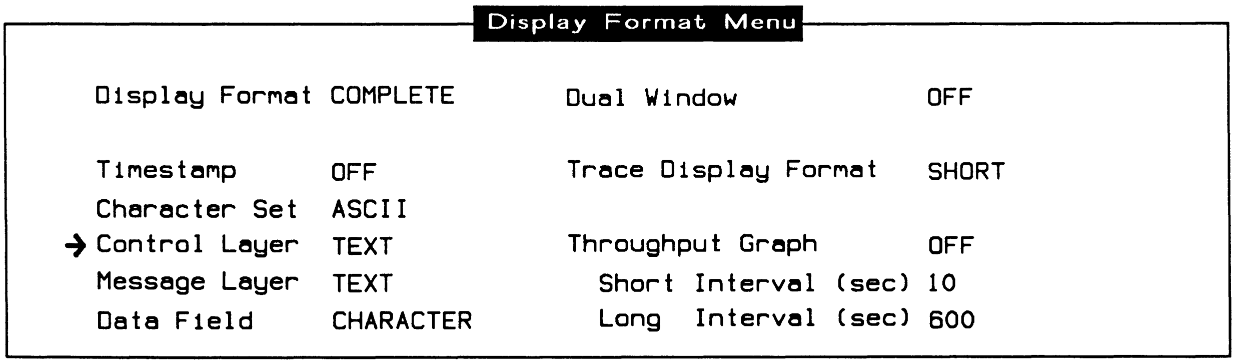


Figure 6–1 BSC Data Flow Diagram – Display and Print

NOTE
Data can only be printed in playback mode.



```
Display Format  COMPLETE        Dual Window             OFF

Timestamp       OFF             Trace Display Format    SHORT
Character Set   ASCII
→ Control Layer TEXT            Throughput Graph        OFF
Message Layer   TEXT              Short Interval (sec)  10
Data Field      CHARACTER         Long  Interval (sec)  600
```

Figure 6–2 Display Format Menu

→ *Display Format*
The default display is short format. *Control Layer, Message Layer,* and *Data Field* can only be modified when *Display Format* is set to *COMPLETE.*

REP_ON (--)
Turns on data display (default).

OFF function key (not highlighted)

REP_OFF (--)
Turns off data display.

OFF function key (highlighted)

REP_COMP (--)
Displays data in a comprehensive report. Each protocol field has its own display generator and thus can be turned on, off, or selected as hex or character display.

COMPLETE function key

REP_SHORT (--)
Displays data in a condensed report (default). This includes the port identifier, cluster controller unit identifier, logical unit device identifier, message type, message length, and the first ten characters of the data field starting at L4–POINTER (see Appendix C). This format is useful for higher speed monitoring as more frames per screen are displayed and processing is kept to a minimum.

SHORT function key

NOTE
The first 20 characters of the message block can be displayed if the 3270_DIS command is issued.

REP_HEX (--)
Displays timestamps or the block sequence number and the port identifier in text. Frame contents are displayed in hex.

HEX function key

REP_CHAR (--)
Displays timestamps or the block sequence number and the port identifier in text. Frame contents are displayed in the currently selected character set.

CHARACTER function key

SPLIT_ON (--)
Displays data in short format with a split screen display. The screen is divided in half with frames sent from the receiver shown on the left and messages sent from the transmitter on the right.

SPLIT function key

NOTE
Only the first 38 characters of a trace statement are displayed when split display format is selected.

SPLIT_OFF (--)
Sets the data display to the full screen short format display (default).

SHORT function key

REP_NONE (--)
Displays only trace statements.

TRACE function key

→ *Timestamp*
Timestamp reporting is available when the display format is not in split mode.

TIME_OFF (--)
Timestamps are not displayed (default). Block sequence numbers are displayed for each received block.

OFF function key

TIME_ON (--)
Displays the start and end of timestamps as minutes, seconds, and tenths of milliseconds. Block sequence numbers for received frames are not displayed.

MM:SS.ssss function key

TIME_DAY (--)

Displays the start and end of timestamps as days, hours, minutes, and seconds. Block sequence numbers are not displayed.

DD HH:MM:SS function key

→ *Character Set*

Selects the character set for data display.

NOTE

Configuring the monitor or emulation to BISYNC ASCII or BISYNC EBCDIC automatically changes the character set.

R=ASCII (--)

Sets the character set for data display to ASCII (default).

ASCII function key

R=EBCDIC (--)

Sets the character set for data display to EBCDIC.

EBCDIC function key

R=HEX (--)

Sets the character set for data display to hex.

HEX function key

R=JIS8 (--)

Sets the character set for data display to JIS8.

JIS8 function key

→ *Control Layer*

CONT_ON (--)

Displays data link control sequences in a detailed report (default).

TEXT function key

CONT_OFF (--)

Data link control sequences are not displayed.

OFF function key

CONT_HEX (--)

Displays data link control sequences in hex.

HEX function key

CONT_CHAR (--)
Displays data link control sequences in the currently selected character set.

CHARACTER function key

→ *Message Layer*
MESS_ON (--)
Displays command and AID (attention identifier) messages in a detailed report (default).

TEXT function key

MESS_OFF (--)
Command and AID messages are not displayed.

OFF function key

MESS_HEX (--)
Displays command and AID messages in hex.

HEX function key

MESS_CHAR (--)
Displays command and AID messages in the currently selected character set.

CHARACTER function key

→ *Data Field*
DATA_ON or **DATA_CHAR** (--)
Displays the data field of messages in the currently selected character set (default).

CHARACTER function key

DATA_OFF (--)
The data field of messages is not displayed.

OFF function key

DATA_HEX (--)
Displays the data field of messages in hex.

HEX function key

3270_ENB (--)
Enables 3270 data stream message procedure reporting.

NOTE
In this reporting procedure, the first four characters of a message block, ending with either an ETX or ETB character in a modified operator, are truncated and are not displayed or included in the message byte length.

3270_DIS (--)
Enables 3780 data stream message procedure reporting. The first 20 characters of the message block are displayed.

CLEAR_CRT (--)
Clears the display in the Data Window.

Display topic
Clear function key

→ *Dual Window*
If two applications have been loaded, the screen can be divided horizontally to display data from both applications. The current application is always displayed in the top window.

FULL (--)
Uses the entire Data Window for the current application.

Dual window commands vary depending on the machine configuration. Table 6-1 shows the relationship between machine configuration, application processors, and dual window commands.

Machine Type	Command	Dual Window AP #	
WAN/WAN	DUAL_1+2	AP #1	AP #2
BRA/WAN	DUAL_1+2	AP #1	AP #2
	DUAL_1+7	AP #1	AP #3
	DUAL_2+7	AP #2	AP #3
PRA	DUAL_3+4	AP #1	AP #2
PRA/BRA/WAN	DUAL_1+2	AP #1	AP #2
	DUAL_1+3	AP #1	AP #4
	DUAL_1+4	AP #1	AP #5
	DUAL_1+7	AP #1	AP #3
	DUAL_2+3	AP #2	AP #4
	DUAL_2+4	AP #2	AP #5
	DUAL_2+7	AP #2	AP #3
	DUAL_3+4	AP #4	AP #5
	DUAL_3+7	AP #4	AP #3
	DUAL_4+7	AP #5	AP #3
BRA/BRA	DUAL_1+2	AP #1	AP #2
	DUAL_1+3	AP #1	AP #4
	DUAL_1+4	AP #1	AP #5
	DUAL_1+5	AP #1	AP #6
	DUAL_1+7	AP #1	AP #3
	DUAL_2+3	AP #2	AP #4
	DUAL_2+4	AP #2	AP #5
	DUAL_2+5	AP #2	AP #6
	DUAL_2+7	AP #2	AP #3
	DUAL_3+4	AP #4	AP #5
	DUAL_3+5	AP #4	AP #6
	DUAL_3+7	AP #4	AP #3
	DUAL_4+5	AP #5	AP #6
	DUAL_4+7	AP #5	AP #3
	DUAL_5+7	AP #6	AP #3
PRA/WAN	DUAL_1+3	AP #1	AP #2
	DUAL_1+4	AP #1	AP #3
	DUAL_3+4	AP #2	AP #3

Table 6-1 Dual Window Commands

→ *Trace Display Format*
Selects the display format for trace statements.

TRACE_SHORT (--)
Displays the trace statement on one line (short format) containing only user-defined text.

SHORT function key

TRACE_COMP (--)
Displays the trace statement on two lines (complete format). Block sequence numbers or timestamps are displayed on the first line, and user-defined text on the second line.

COMPLETE function key

→ *Throughput Graph*
The throughput rate can be calculated, displayed as a bar graph, and printed out. The BSC monitor calculates throughput by counting the number of bytes on each side of the line during two intervals - one short, one long. This figure is divided by the time interval to arrive at a bits per second figure for each time interval (for both Tx and Rx data).

NOTE
For accurate throughput measurement, the bit rate (line speed) must be set on the Monitor/Emulation Configuration Menu or in the INTERFACE-SPEED variable to match the actual line speed.

The baud rate, as stored in the variable INTERFACE-SPEED, is used to calculate a percentage throughput based on theoretical limits.

INTERFACE-SPEED (--address)
Contains the current bit rate (default value is 9600).

Example:
Set the throughput measurement speed to 2400.
```
2400 INTERFACE-SPEED !
TPR_ON
```

TPR_ON (--)
Calculates and displays the throughput rate as a bar graph.

DISPLAY function key

WARNING
If the short interval, long interval, or speed is changed, TPR_ON must be called after the changes are made.

TPR_OFF (--)
The throughput rate is not calculated or displayed (default).

OFF function key

PRINT_TPR (--)
Calculates and displays the throughput rate as a bar graph, and prints the long term interval measurements.

DISPLAY AND PRINT function key

→ *Short Interval*
Sets the short time interval, in seconds, for measuring, displaying, and printing the throughput results.

SHORT-INTERVAL (--address)
Contains the current duration of the short interval (default value is 10 seconds).

Example:
Set the short interval to 20 seconds.
```
20 SHORT-INTERVAL !
TPR_ON
```

Modify Short Interval function key

→ *Long Interval*
Sets the long time interval, in seconds, for measuring, displaying, and printing the throughput results.

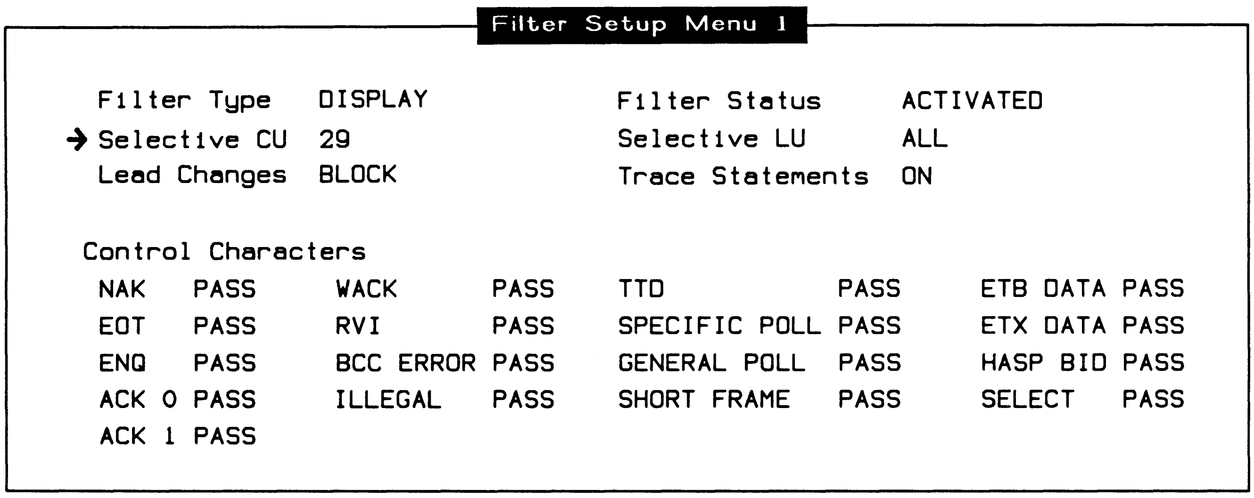
LONG-INTERVAL (--address)
Contains the current duration of the long interval (default value is 600 seconds).

Example:
Set the long interval to 300 seconds.
```
300 LONG-INTERVAL !          ( Set long interval )
TPR_ON                       ( Turn on throughput measurement )
```

Modify Long Interval function key

7 FILTERS

Filters provide the capability of passing or blocking specific events from the display, capture RAM, or disk recording. These three filters act independently. This section describes the commands used to set up each of the three filters.



```
                              Filter Setup Menu 1

   Filter Type    DISPLAY               Filter Status       ACTIVATED
 → Selective CU   29                    Selective LU        ALL
   Lead Changes   BLOCK                 Trace Statements    ON

  Control Characters
   NAK    PASS    WACK       PASS    TTD            PASS    ETB DATA PASS
   EOT    PASS    RVI        PASS    SPECIFIC POLL  PASS    ETX DATA PASS
   ENQ    PASS    BCC ERROR  PASS    GENERAL POLL   PASS    HASP BID PASS
   ACK 0  PASS    ILLEGAL    PASS    SHORT FRAME    PASS    SELECT   PASS
   ACK 1  PASS
```

Figure 7–1 Filter Setup Menu 1

WARNING
On Filter Setup Menu 1 only, Selective CU is referred to as the attached logical unit device. Outside of this menu, LU is referred to as the logical unit device and CU is referred to as the cluster controller (i.e. LogicalDev and LogicalUnit variables).

→ *Filter Type*
There are three separate filter processes which act independently of each other: *DISPLAY, RAM,* and *DISK*.

→ *Selective CU*
Messages, initiated between either *all* or a *selected* logical unit device(s) and the host system, can be passed to capture RAM, disk, or display.

NOTE
The selected logical unit device must be the one attached to the last cluster controller. Selection changes on Filter Setup Menu 1 and 2 affect only the currently displayed logical unit device (Selective CU) and cluster controller unit (Selective LU), as specified on Filter Setup Menu 1.

R–CU (--)
Passes messages, initiated between all active logical unit devices and the host system, to the display.

→ *Filter Type*
DISPLAY function key
→ *Selective CU*
ALL function key

R–SEL–CU (--address)
Contains the selective logical unit device for display filters. Valid values are 0 through 31.

Example:
Set the display filter to only pass events or messages initiated between logical unit device 8 and the host system.

```
8 R-SEL-CU !        ( Selective CU is set to logical device number 8 )
R+CU
```

→ *Filter Type*
DISPLAY function key
→ *Selective CU*
ONE function key

R+CU (--)
Passes messages, initiated between the selected logical unit device and the host system, to the display.

C–CU (--)
Passes messages, initiated between all active logical unit devices and the host system, to capture RAM.

→ *Filter Type*
RAM function key
→ *Selective CU*
ALL function key

C−SEL−CU (--address)
Contains the selective logical unit device for capture RAM filters. Valid values are 0 through 31.

Example:
Set the capture RAM filter to only pass events or messages initiated between logical unit device 3 and the host system.

```
3 C-SEL-CU !         ( Selective CU is set to logical device number 3 )
C+CU
```

→ *Filter Type*
RAM function key
→ *Selective CU*
ONE function key

C+CU (--)
Passes messages, initiated between the selected logical unit device and the host system, to capture RAM.

D−CU (--)
Passes messages, initiated between all active logical unit devices and the host system, to disk.

→ *Filter Type*
DISK function key
→ *Selective CU*
ALL function key

D−SEL−CU (--address)
Contains the selective logical unit device for disk recording filters. Valid values are 0 through 31.

Example:
Set the disk filter to only pass events or messages initiated between logical unit device 8 and the host system.

```
8 D-SEL-CU !         ( Selective CU is set to logical device number 8 )
D+CU
```

→ *Filter Type*
DISK function key
→ *Selective CU*
ONE function key

D+CU (--)
Passes messages, initiated between the selected logical unit device and the host system, to disk.

→ *Lead Changes*
Lead changes can be blocked (default) or passed.

R+LEADS (--)
Passes lead changes to the display.

→ *Filter Type*
DISPLAY function key
→ *Lead Changes*
PASS function key

R–LEADS (--)
Blocks lead changes from the display.

→ *Filter Type*
DISPLAY function key
→ *Lead Changes*
BLOCK function key

C+LEADS (--)
Passes lead changes to capture RAM.

→ *Filter Type*
RAM function key
→ *Lead Changes*
PASS function key

C–LEADS (--)
Blocks lead changes from capture RAM.

→ *Filter Type*
RAM function key
→ *Lead Changes*
BLOCK function key

D+LEADS (--)
Passes lead changes to disk.

→ *Filter Type*
DISK function key
→ *Lead Changes*
PASS function key

D–LEADS (--)
Blocks lead changes from disk.

→ *Filter Type*
DISK function key
→ *Lead Changes*
BLOCK function key

→ *Filter Status*
Filters can be deactivated (default) or activated at any time. When the filter status is changed, the connection diagram changes to reflect this. Figure 7–2 shows live data as the data source with display filters activated. If deactivated, all lead changes, trace statements, and messages go to display.

NOTE
Filters can only be activated/deactivated via function keys.

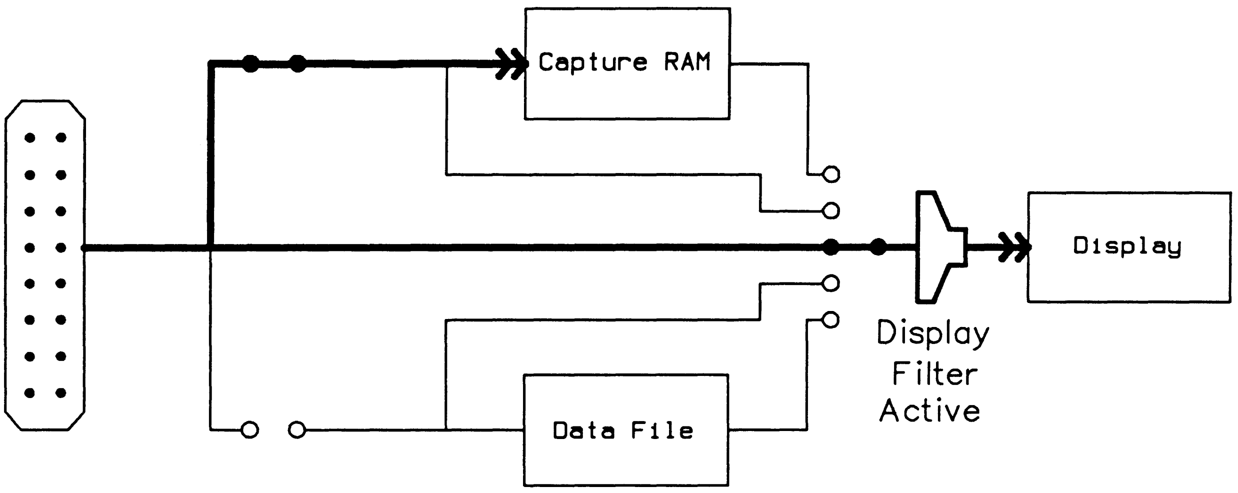


Figure 7–2 Connection Diagram – Display Filters Activated

→ *Selective LU*
Messages, initiated between either *all* or a *selected* cluster controller(s) and the host system, can be passed to capture RAM, disk, or display.

R–LU (--)
Passes messages, initiated between all active cluster controllers and the host system, to the display.

→ *Filter Type*
DISPLAY function key
→ *Selective LU*
ALL function key

R–SEL–LU (--address)

Contains the selective cluster controller for display filters. Valid values are 0 through 31.

Example:
Set the display filter to only pass events or messages initiated between cluster controller 8 and the host system.

```
8 R-SEL-LU !          ( Selective LU is set to cluster controller 8 )
R+LU
```

→ *Filter Type*
DISPLAY function key
→ *Selective LU*
ONE function key

R+LU (--)

Passes messages, initiated between the selected cluster controller and the host system, to the display.

C–LU (--)

Passes messages, initiated between all active cluster controllers and the host system, to capture RAM.

→ *Filter Type*
RAM function key
→ *Selective LU*
ALL function key

C–SEL–LU (--address)

Contains the selective cluster controller for capture RAM filters. Valid values are 0 through 31.

Example:
Set the capture RAM filter to only pass events or messages initiated between cluster controller 8 and the host system.

```
8 C-SEL-LU !          ( Selective LU is set to cluster controller 8 )
C+LU
```

→ *Filter Type*
RAM function key
→ *Selective LU*
ONE function key

C+LU (--)

Passes messages, initiated between the selected cluster controller and the host system, to capture RAM.

D–LU (--)
Passes messages, initiated between all active cluster controllers and the host system, to disk.

→ *Filter Type*
DISK function key
→ *Selective LU*
ALL function key

D–SEL–LU (--address)
Contains the selective cluster controller for disk recording filters. Valid values are 0 through 31.

Example:
Set the disk filter to only pass events or messages initiated between cluster controller 8 and the host system.

```
8 D-SEL-LU !          ( Selective LU is set to cluster controller 8 )
D+LU
```

→ *Filter Type*
DISK function key
→ *Selective LU*
ONE function key

D+LU (--)
Passes messages, initiated between the selected cluster controller and the host system, to disk.

→ *Trace Statements*
Trace statements can be blocked or passed (default).

YES RTRACE (--)
Passes trace statements to the display.

→ *Filter Type*
DISPLAY function key
→ *Trace Statements*
ON function key

NO RTRACE (--)
Blocks trace statements from the display.

→ *Filter Type*
DISPLAY function key
→ *Trace Statements*
OFF function key

YES CTRACE (--)
Passes trace statements to capture RAM.

→ *Filter Type*
RAM function key
→ *Trace Statements*
ON function key

NO CTRACE (--)
Blocks trace statements from capture RAM.

→ *Filter Type*
RAM function key
→ *Trace Statements*
OFF function key

YES DTRACE (--)
Passes trace statements to disk.

→ *Filter Type*
DISK function key
→ *Trace Statements*
ON function key

NO DTRACE (--)
Blocks trace statements from disk.

→ *Filter Type*
DISK function key
→ *Trace Statements*
OFF function key

Control Characters
Data link control events can be blocked or passed (default).

NOTE
Commands for display filters and NAK control characters are described here as an example. For a complete list of commands, see Table 7–1.

R=ALL (--)
Passes all message types (i.e. data link control sequences, command codes, and attention identifiers) to the display.

→ *Filter Type*
DISPLAY function key
→ *NAK*
ALL FRAMES function key

R=NONE (--)
Blocks all message types (i.e. data link control sequences, command codes, and attention identifiers) from the display.

→ *Filter Type*
DISPLAY function key
→ *NAK*
NO FRAMES function key

R+NAK (--)
Passes negative acknowledgements to the display.

→ *Filter Type*
DISPLAY function key
→ *NAK*
PASS function key

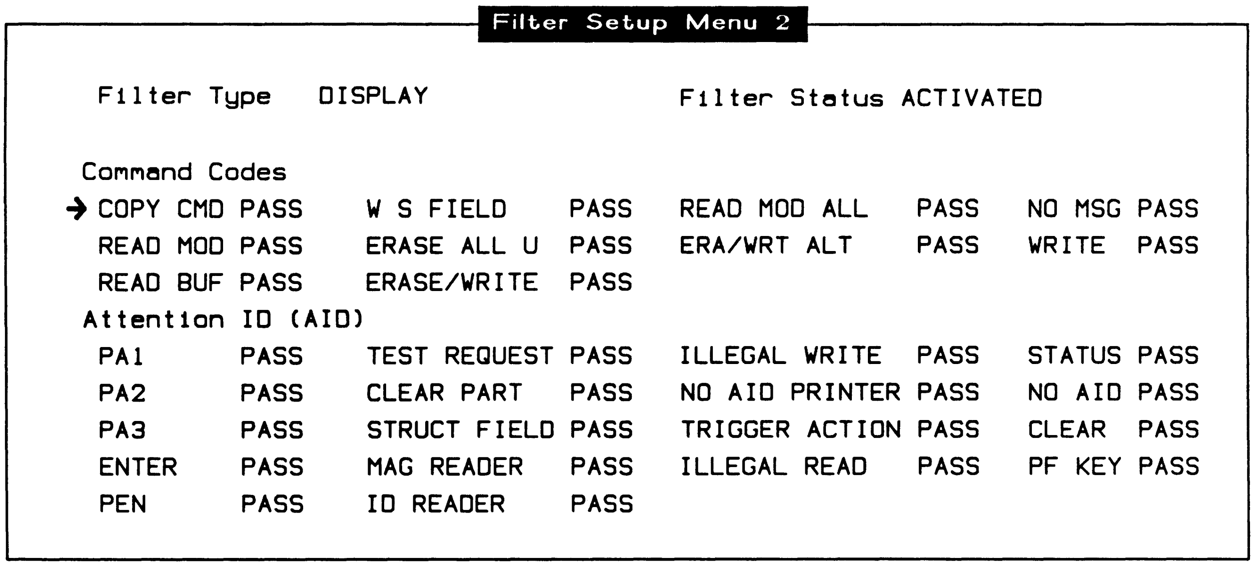
R−NAK (--)
Blocks negative acknowledgements from the display.

→ *Filter Type*
DISPLAY function key
→ *NAK*
BLOCK function key

The following table lists the corresponding commands to pass/block data link control sequences to/from display, capture RAM, or disk.

Description		Display	RAM	Disk
Negative Acknowledgement	Pass	R+NAK	C+NAK	D+NAK
	Block	R-NAK	C-NAK	D-NAK
End of Transmission	Pass	R+EOT	C+EOT	D+EOT
	Block	R-EOT	C-EOT	D-EOT
Enquiry	Pass	R+ENQ	C+ENQ	D+ENQ
	Block	R-ENQ	C-ENQ	D-ENQ
Acknowledgement 0	Pass	R+ACK0	C+ACK0	D+ACK0
	Block	R-ACK0	C-AKC0	D-ACK0
Acknowledgement 1	Pass	P+ACK1	C+ACK1	D+ACK1
	Block	R-ACK1	C-ACK1	D-ACK1
Wait for Positive Acknowledgement	Pass	R+WACK	C+WACK	D+WACK
	Block	R-WACK	C-WACK	D-WACK
Reverse Interrupt	Pass	R+RVI	C+RVI	D+RVI
	Block	R-RVI	C-RVI	D-RVI
Block Check Character Errors	Pass	R+BCC_ERROR	C+BCC_ERROR	D+BCC_ERROR
	Block	R-BCC_ERROR	C-BCC_ERROR	D-BCC_ERROR
Illegal	Pass	R+ILLEGAL	C+ILLEGAL	D+ILLEGAL
	Block	R-ILLEGAL	C-ILLEGAL	D-ILLEGAL
Temporary Text Delay	Pass	R+TTD	C+TTD	D+TTD
	Block	R-TTD	C-TTD	D-TTD
Specific Poll	Pass	R+S_POLL	C+S_POLL	D+S_POLL
	Block	R-S_POLL	C-S_POLL	D-S_POLL
General Poll	Pass	R+G_POLL	C+G_POLL	D+G_POLL
	Block	R-G_POLL	C-G_POLL	D-G_POLL
Short Frame	Pass	R+SHORT	C+SHORT	D+SHORT
	Block	R-SHORT	C-SHORT	D-SHORT
End of Transmission Block	Pass	R+ETB_DATA	C+ETB_DATA	D+ETB_DATA
	Block	R-ETB_DATA	C-ETB_DATA	D-ETB_DATA
End of Text	Pass	R+ETX_DATA	C+ETX_DATA	D+ETX_DATA
	Block	R-ETX_DATA	C-ETX_DATA	D-ETX_DATA
HASP Bid	Pass	R+HASP_BID	C+HASP_BID	D+HASP_BID
	Block	R-HASP_BID	C-HASP_BID	D-HASP_BID
Select	Pass	R+SELECT	C+SELECT	D+SELECT
	Block	R-SELECT	C-SELECT	D-SELECT

Table 7–1 Data Link Control Sequence Filter Commands



```
                        Filter Setup Menu 2

   Filter Type   DISPLAY                 Filter Status ACTIVATED

  Command Codes
→ COPY CMD PASS    W S FIELD    PASS   READ MOD ALL   PASS    NO MSG PASS
  READ MOD PASS    ERASE ALL U  PASS   ERA/WRT ALT    PASS    WRITE  PASS
  READ BUF PASS    ERASE/WRITE  PASS
  Attention ID (AID)
  PA1      PASS    TEST REQUEST PASS   ILLEGAL WRITE  PASS    STATUS PASS
  PA2      PASS    CLEAR PART   PASS   NO AID PRINTER PASS    NO AID PASS
  PA3      PASS    STRUCT FIELD PASS   TRIGGER ACTION PASS    CLEAR  PASS
  ENTER    PASS    MAG READER   PASS   ILLEGAL READ   PASS    PF KEY PASS
  PEN      PASS    ID READER    PASS
```

Figure 7–3 Filter Setup Menu 2

Command Codes

The following table lists the corresponding commands to pass/block command codes to/from display, capture RAM, or disk.

Description		Display	RAM	Disk
Copy	Pass	P+COPY_CMD	C+COPY_CMD	D+COPY_CMD
	Block	R-COPY_CMD	C-COPY_CMD	D-COPY_CMD
Read Modified	Pass	R+READ_MOD	C+READ_MOD	D+READ_MOD
	Block	R-READ_MOD	C-READ_MOD	D-READ_MOD
Read Buffer	Pass	R+READ_BUF	C+READ_BUF	D+READ_BUF
	Block	R-READ_BUF	C-READ_BUF	D-READ_BUF
Write Structure Field	Pass	R+W_S_FIELD	C+W_S_FIELD	D+W_S_FIELD
	Block	R-W_S_FIELD	C-W_S_FIELD	D-W_S_FIELD
Erase All Unprotected	Pass	R+ERASE_ALL_U	C+ERASE_ALL_U	D+ERASE_ALL_U
	Block	R-ERASE_ALL_U	C-ERASE_ALL_U	D-ERASE_ALL_U
Erase/Write	Pass	R+ERASE/WRITE	C+ERASE/WRITE	D+ERASE/WRITE
	Block	R-ERASE/WRITE	C-ERASE/WRITE	D-ERASE/WRITE
Read Modified All	Pass	R+READ_MOD_ALL	C+READ_MOD_ALL	D+READ_MOD_ALL
	Block	R-READ_MOD_ALL	C-READ_MOD_ALL	D-READ_MOD_ALL
Erase/Write Alternate	Pass	R+ERA/WRT_ALT	C+ERA/WRT_ALT	D+ERA/WRT_ALT
	Block	R-ERA/WRT_ALT	C-ERA/WRT_ALT	D-ERA/WRT_ALT
Chain Messages	Pass	R+NO_MSG	C+NO_MSG	D+NO_MSG
	Block	R-NO_MSG	C-NO_MSG	D-NO_MSG
Write	Pass	R+WRITE	C+WRITE	D+WRITE
	Block	R-WRITE	C-WRITE	D-WRITE

Table 7-2 Command Code Filter Commands

Attention ID (AID)

The following table lists the corresponding commands to pass/block attention identifiers to/from display, capture RAM, or disk.

Description		Display	RAM	Disk
PA1	Pass	R+PA1	C+PA1	D+PA1
	Block	R−PA1	C−PA1	D−PA1
PA2	Pass	R+PA2	C+PA2	D+PA2
	Block	R−PA2	C−PA2	D−PA2
PA3	Pass	R+PA3	C+PA3	D+PA3
	Block	R−PA3	C−PA3	D−PA3
Enter	Pass	R+ENTER	C+ENTER	D+ENTER
	Block	R−CENTER	C−CENTER	D−CENTER
Pen	Pass	R+PEN	C+PEN	D+PEN
	Block	R−PEN	C−PEN	D−PEN
Test Request	Pass	R+TEST_REQ	C+TEST_REQ	D+TEST_REQ
	Block	R−TEST_REQ	C−TEST_REQ	D−TEST_REQ
Clear Partition	Pass	R+CLEAR_PART	C+CLEAR_PART	D+CLEAR_PART
	Block	R−CLEAR_PART	C−CLEAR_PART	D−CLEAR_PART
Structure Fields	Pass	R+S_FIELD	C+S_FIELD	D+S_FIELD
	Block	R−S_FIELD	C−S_FIELD	D−S_FIELD
MAG Reader	Pass	R+MAG_READER	C+MAG_READER	D+MAG_READER
	Block	R−MAG_READER	C−MAG_READER	D−MAG_READER
ID Reader	Pass	R+ID_READER	C+ID_READER	D+ID_READER
	Block	R−ID_READER	C−ID_READER	D−ID_READER
Illegal Write	Pass	R+ILL_WRITE	C+ILL_WRITE	D+ILL_WRITE
	Block	R−ILL_WRITE	C−ILL_WRITE	D−ILL_WRITE
Printer	Pass	R+NO_AID_P	C+NO_AID_P	D+NO_AID_P
	Block	R−NO_AID_P	C−NO_AID_P	D−NO_AID_P
Trigger Action	Pass	R+TRIG_ACTION	C+TRIG_ACTION	D+TRIG_ACTION
	Block	R−TRIG_ACTION	C−TRIG_ACTION	D−TRIG_ACTION
Illegal Read	Pass	R+ILL_READ	C+ILL_READ	D+ILL_READ
	Block	R−ILL_READ	C−ILL_READ	D−ILL_READ
Status	Pass	R+STATUS	C+STATUS	D+STATUS
	Block	R−STATUS	C−STATUS	D−STATUS
No AID	Pass	R+NO_AID	C+NO_AID	D+NO_AID
	Block	R−NO_AID	C−NO_AID	D−NO_AID
Clear	Pass	R+CLEAR	C+CLEAR	D+CLEAR
	Block	R−CLEAR	C−CLEAR	D−CLEAR
All PF Key	Pass	R+PF_KEY	C+PF_KEY	D+PF_KEY
	Block	R−PF_KEY	C−PF_KEY	D−PF_KEY

Table 7–3 Attention Identifier Filter Commands

8 DECODE

This section describes the data flow diagram and lists the variables in which decoded information is saved.

Data or lead changes from the interface, capture RAM, or disk file are decoded. Decoded information is stored in a pool of variables for later use by either a test program or other parts of the monitor application.

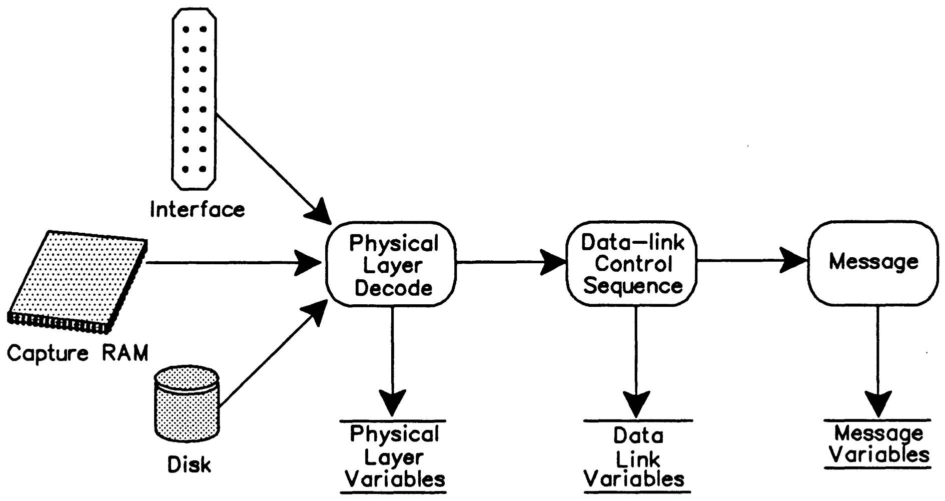


Figure 8–1 BSC Data Flow Diagram – Decode

8.1 Communication Variables

The following variables are set during the decode process. They contain protocol specific information about the last data or control message blocks processed by the monitor as defined by IBM 3270 Information Display System: 3274 Control Unit Description and Programmer's Guide, GA23-0061-2. publication.

NOTE
These variables can be read using the @ (fetch) operation.

Physical Layer

The physical layer decode operation saves information concerning message length, timestamps, port identifier, and block sequence number. For lead transitions, information is saved concerning the changed lead(s); for timers, the number of the expired timer.

PORT-ID (--address)
Contains a 2 byte value indentifying received direction for data. The lower byte indicates the TX (hex value 08) or RX (hex value 20) receive stream. The upper byte indicates the application processor that received the frame.

Example:
Determine the direction of the received stream.

```
PORT-ID @
OXFF AND                         ( The AND operation eliminates the upper byte )
```

This operation leaves the received stream direction on the stack. It is 0 for a trace statement, or equal to one of the following pre-defined constants: TO_DTE_RX for TX data or TO_DCE_RX for RX data. For further explanation of port identification, consult the Programmer's Reference Manuals.

START-TIME (--address)
Contains the 48 bit start of frame timestamp for data. Use with the GET_TSTAMP_MILLI or GET_TSTAMP_MICRO commands. See the Programmer's Reference Manual.

Example:
Obtain the start of message timestamp including year, month, day, hour, minute, second, and millisecond.

```
START-TIME GET_TSTAMP_MILLI
```

NOTE
The @ (fetch) operation is not performed. Seven values are left on the stack as described in the Programmer's Reference Manual.

END-TIME (--address)
Contains the 48 bit end of message timestamp for data. Use with the GET_TSTAMP_MILLI or GET_TSTAMP_MICRO commands. See the START-TIME example.

BLOCK-COUNT (--address)
Contains the sequential block sequence number for live data. Every received message is assigned a unique sequence number. Each side, Tx or Rx, maintains a separate set of sequence numbers. Initially contains a value of zero and is incremented by one each time a new block is received.

REC-LENGTH (-- address)
Contains the length of the last received message block. This does not include the PAD, SYN, or CRC bytes.

REC–POINTER (-- address)

Contains the pointer to the first byte in the last received block. Since this variable contains the address of the first byte, a double fetch operation is necessary to obtain message contents.

Example:
Obtain the second byte of the received message.

```
REC-POINTER @  1+ C@
```

NOTE
The @ command gets the address of the first byte in the received block. This first value is then incremented by one and one byte is fetched from the resulting address.

STATUS_ERR? (-- flag)

Returns true if an error is detected in the currently processed frame. The following commands return true if that particular error occurred.

Command	Error Type
OVERRUN_ERR?	Receiver overrun
CRC_ERR?	CRC error
LONG_FRM_ERR?	Frame is longer than supported by operating system buffers
PARITY_ERR?	Parity error

LEAD–NUMBER (--address)

Contains the received lead identifier used in the test manager.

TIMER–NUMBER (--address)

Contains the identifier of the expired timer. Valid values are 1 through 128.

Logical Units

LogicalUnit (--address)

Contains the CU (cluster controller) identifier of the last received message block. Valid identifiers are 0 through 31.

LogicalDev (--address)

Contains the LU (logical unit device) unit identifier of the last received message block. Valid identifiers are 0 through 31.

NOTE
All received messages must be preceded by a cluster/device address header.

Pointers and Lengths

Refer to Appendix C for illustrations of pointers and lengths for various types of messages.

L2–LENGTH (--address)
Contains the length of the received message block. This does not include the PAD, SYN, or CRC byte.

NOTE
L2–LENGTH should not be used in a test script.

L2–POINTER (--address)
Contains the pointer to the first byte of the last received message block. Since this variable contains the address of the first byte, a double fetch operation is necessary to obtain the message content.

Example:
Obtain the third byte of the received message.

```
L2-LENGTH @ 2+ C@
```

The @ (fetch) command retrieves the address of the first byte of the received message. This first value is then incremented by two and one byte is fetched from the resulting address.

NOTE
L2–POINTER should not be used in a test script.

L3–POINTER (--address)
Contains the pointer to the beginning of the header field of the last received message block.

NOTE
This pointer is set when the FRAME–ID value is ETX_DATA, ETB_DATA, or ILLEGAL. Valid FRAME–ID values are listed in Table 12–1.

L3–LENGTH (--address)
Contains the length of the header field of the last received message block.

NOTE
This length is set when the FRAME–ID value is ETX_DATA, ETB_DATA, or ILLEGAL. Valid FRAME–ID values are listed in Table 12–1.

L4–POINTER (--address)
Contains the pointer to the beginning of the orders and/or data field of the last received message block.

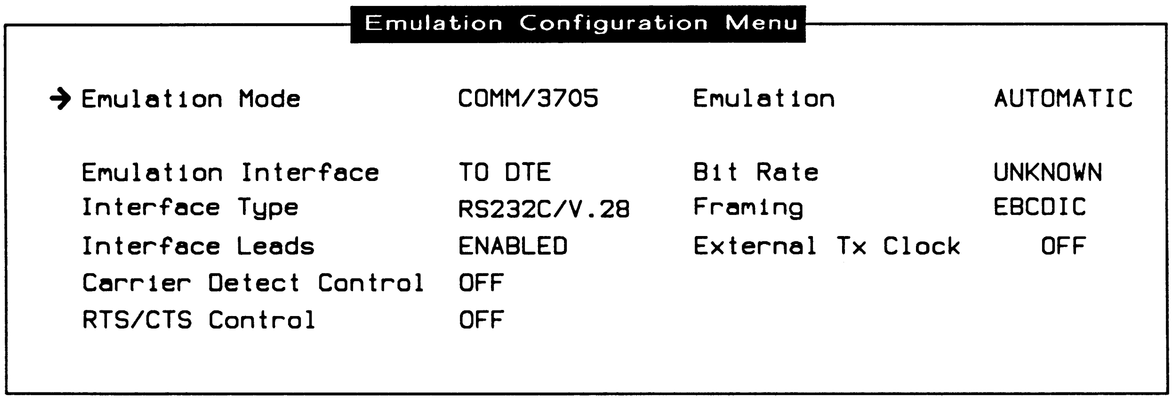
NOTE
This pointer is set when an order or data message is received, as described in PACKET–ID. Valid PACKET–ID values are listed in Table 12–2.

L4–LENGTH (--address)
Contains the length of the orders and/or data field of the last received message block.

NOTE
This length is set when an order or data message, as described in PACKET–ID, is received. Valid PACKET–ID values are listed in Table 12–2.

9
EMULATION CONFIGURATION

This section displays the Emulation Configuration Menu and Device Setup Menu and describes commands corresponding to each item.



```
                    Emulation Configuration Menu

→ Emulation Mode          COMM/3705      Emulation           AUTOMATIC

  Emulation Interface     TO DTE         Bit Rate            UNKNOWN
  Interface Type          RS232C/V.28    Framing             EBCDIC
  Interface Leads         ENABLED        External Tx Clock     OFF
  Carrier Detect Control  OFF
  RTS/CTS Control         OFF
```

Figure 9–1 Emulation Configuration Menu

The emulation is active at all times i.e automatic responses to protocol events are generated even during parameter setup. However, most of the commands which change the physical layer require re–initialization of the emulation program. This is accomplished using the RECONFIGURE command.

RECONFIGURE (--)
Initializes the BSC emulation and configures the physical interface.

NOTE
Use RECONFIGURE once after all physical layer changes are made.

→ *Emulation Mode*
Selects whether to emulate a CLST/3274 (cluster controller) or a COMM/3705 (communications controller).

COMM/3705 (--)
Selects the COMM/3705 state machine emulation (default) and forces the emulation into the multipoint POLL_MP state. See Table 11-1 for state descriptions.

COMM/3705 function key

NOTE
Polling processes and associated timers are not activated until the POLL_ON command is issued.

CLST/3274 (--)
Selects the CLST/3274 state machine emulation and forces the emulation into the multipoint IDLE_CC state. See Table 11-2 for state descriptions.

CLST/3274 function key

→ *Emulation*
EMUL_OFF (--)
Disables the automatic emulation; the SDLC state machine does not transmit any frames without manual or test program intervention.

MANUAL function key

EMUL_ON (--)
Enables the automatic emulation; the SDLC state machine responds to all incoming frames according to SDLC protocol.

AUTOMATIC function key

→ *Emulation Interface*
Selects the physical type of emulation.

NOTE
Refer to Table 9-1 for clocking selections depending on the emulation interface.

TO_DCE_IF (--)
Selects the 'to DCE' interface.

TO DCE function key

TO_DTE_IF (--)
Selects the 'to DTE' interface (default). The tester supplies all necessary clocking information to the interface connector.

TO DTE function key

→ *Interface Type*

IF=V28 (--)
Selects the V.28/RS-232C connector (default) and electrically isolates the other connectors on the port.

RS232C/V.28 function key

IF=V11 (--)
Selects the V.11/X.21 connector and electrically isolates the other connectors on the port.

RS422/V.11 function key

IF=V35 (--)
Selects the V.35 connector and electrically isolates the other connectors on the port.

V.35 function key

IF=V36 (--)
Selects the V.36 (RS-449) connector and electrically isolates the other connectors on the port.

RS449/V.36 function key

NOTE
A WAN tester has a V.28, V.11, and either a V.35 or V.36 connector. These commands are only applicable if the program is running on a WAN interface.

→ *Interface Leads*

Individual or all interface leads can be enabled or disabled (default). Leads must be enabled for test manager detection.

ENABLE_LEAD (lead identifier --)
Enables the specified lead. Refer to the Programmer's Reference Manual for a list of supported leads for each interface type.

Example:
Enable the request to send lead.
```
IRS ENABLE_LEAD
```

DISABLE_LEAD (lead identifier --)
Disables (default) the specified lead. Refer to the Programmer's Reference Manual for a list of supported leads for each interface type.

Example:
Disable the clear to send lead.
```
ICS DISABLE_LEAD
```

ALL_LEADS (-- lead identifier)
Enables/disables all leads supported on the currently selected WAN interface. ALL_LEADS must be used with ENABLE_LEAD or DISABLE_LEAD.

Example 1:
Enable all leads for the current interface.
```
ALL_LEADS ENABLE_LEAD
```

ENABLED function key

Example 2:
Disable all leads for the current interface.
```
ALL_LEADS DISABLE_LEAD
```

DISABLED function key

→ *Carrier Detect Control*
Selects whether the DCE device asserts CD (carrier detect) leads to indicate valid transmission.

CD_ENB (--)
Enables CD control leads.

ON function key

CD_DIS (--)
Disables CD control leads (default).

OFF function key

→ *RTS/CTS Control*
Selects whether the DTE device asserts RTS (request to send) leads and waits for an asserted CTS (clear to send) lead before transmitting data.

RTS–IGNORE (--address)
Contains a flag which enables/disables the emulation to respond to an inquiry regardless of the RTS lead condition.

Example 1:
Set the emulation to respond to all inquiries regardless of the RTS lead condition.
```
YES  RTS-IGNORE  !
```

Example 2:
Set the emulation to wait for the RTS lead to turn on before responding.
```
NO  RTS-IGNORE  !   ( Default )
```

R/CTS_ENB (--)
Enables RTS/CTS control leads.

ON function key

R/CTS_DIS (--)
Disables RTS/CTS control leads (default).

OFF function key

NOTE
The RTS lead is turned off after transmission. This handshaking is ignored if the CTS lead is asserted permanently by the test partner.

→ *Bit Rate*
The interface speed can be selected from preset values on the Interface Port Speed Menu, set to a user-defined speed, or measured depending on the emulation interface and clocking selections.

Clocking	TO DTE	TO DCE
OFF	Select	Measure
ON	Measure	Select

Table 9–1 Effect of Clocking and Emulation Interface Selections on Bit Rate

NOTE
Clocking is provided by the attached equipment when the bit rate can be selected.

INTERFACE–SPEED (--address)
Contains the current bit rate (default is 9600).

Example:
Set the interface speed to 2400.
```
2400  INTERFACE-SPEED !
```

NOTE
Integer values must be written to INTERFACE–SPEED. Thus, to obtain a bit rate of 134.5, either 134 or 135 can be written to INTERFACE–SPEED.

→ *Framing*
BSC–EBCDI (--)
Uses EBCDIC framing (default) with the following characteristics:
- NRZ (standard) clocking
- 8 bits per character
- No parity
- EBCDIC character set
- Sync character of hex 32
- Interframe fill is mark
- Rest idle is mark
- DCD control is on
- CRC calculation according to CRC–16
- Strip sync is on

EBCDIC function key

BSC−ASCII (--)
Uses ASCII framing with the following characteristics:
- NRZ (standard) clocking
- 7 bits per character
- odd parity
- ASCII character set
- Sync character of hex 16
- Interframe fill is mark
- Rest idle is mark
- DCD control is on
- CRC calculation according to VRC/LRC
- Strip sync is on

ASCII function key

→ *External Tx Clock*
EXT_CLOCK (--)
Clocking is provided by the DTE.

ON function key

STD_CLOCK (--)
Clocking is provided by the DCE (default).

OFF function key

The following sequence illustrates the use of the configuration commands. RECONFIGURE is only called at the end of the configuration sequence.

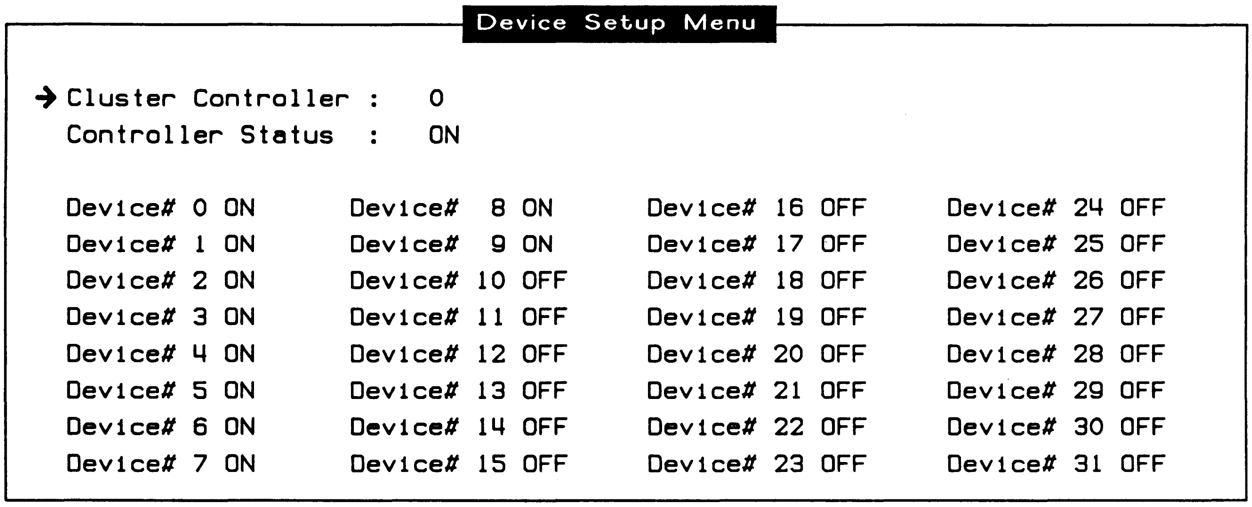
```
TCLR
IF=V35                      ( Use V.35 test connector )
COMM/3705                   ( TCU simulation )
TO_DTE_IF                   ( Set for DCE emulation )
2400 INTERFACE-SPEED !      ( Set baud rate )
CD_ENB                      ( CD control leads on )
R/CTS_ENB                   ( RTS/CTS control leads on )
STD_CLK                     ( Use internal clock )
RECONFIGURE                 ( Configure emulation )
```

9.1 Device Setup

When emulating a 3705 communications controller, the Bisync Emulation can communicate with up to 32 cluster controllers, each having up to 32 devices. Selecting specific clusters sets up the poll train list for the emulator (i.e. the emulation sends a general poll to each cluster controller selected in the menu). Any devices not responding are retried two times prior to polling the next device.

The CLST/3274 Bisync Emulation can emulate up to 32 cluster controllers simultaneously. Selecting a cluster and device ensures a response to a specific poll to that device. The selected cluster responds to a general poll if it is activated and not busy.

NOTE
Refer to Appendix B for device addressing.



```
                              Device Setup Menu

→ Cluster Controller  :   0
  Controller Status   :   ON

  Device# 0 ON      Device#  8 ON     Device# 16 OFF     Device# 24 OFF
  Device# 1 ON      Device#  9 ON     Device# 17 OFF     Device# 25 OFF
  Device# 2 ON      Device# 10 OFF    Device# 18 OFF     Device# 26 OFF
  Device# 3 ON      Device# 11 OFF    Device# 19 OFF     Device# 27 OFF
  Device# 4 ON      Device# 12 OFF    Device# 20 OFF     Device# 28 OFF
  Device# 5 ON      Device# 13 OFF    Device# 21 OFF     Device# 29 OFF
  Device# 6 ON      Device# 14 OFF    Device# 22 OFF     Device# 30 OFF
  Device# 7 ON      Device# 15 OFF    Device# 23 OFF     Device# 31 OFF
```

Figure 9–2 Device Setup Menu

→ *Cluster Controller*
Specifies the cluster controller that is activated or deactivated in COMM/3705 and CLST/3274 emulation. Valid values are 0 through 31.

NOTE
Refer to the ACT and DEACT commands to specify a cluster controller.

→ *Controller Status*
DEACT (cluster--)

In CLST/3274 emulation, deactivates the specified cluster controller.

In COMM/3705 emulation, configures the poll train array to ignore the specified cluster controller.

Example 1:
In CLST/3274 emulation, deactivate cluster controller 5.

```
5 DEACT
```

Example 2:
In COMM/3705 emulation, configure the poll train array to ignore cluster controller 6.

```
6 DEACT
```

OFF function key

→ *Device# n*
ACT (-1 \ dev1 \ dev2 \ ...... \ cluster--)

In CLST/3274 emulation, activates the specified cluster controller and the associated logical unit devices.

In COMM/3705 emulation, configures the poll train array to recognize the specified cluster controller and the associated logical unit devices.

Example 1:
In COMM/3705 emulation, configure the poll train array to recognize cluster controller 5 and logical unit devices 0, 3, and 10.

```
-1 0 3 10 5 ACT
```

Example 2:
In CLST/3274 emulation, activate cluster controller 5 and associated logical unit devices 0, 3, and 10.

```
-1 0 3 10 5 ACT
```

ON function key

NOTE
In COMM/3705 emulation, up to 32 cluster controllers and the associated 32 logical unit devices can be recognized. In CLST/3274 emulation, 32 cluster controller units and the associated 32 logical unit devices can be activated.

SET_TRANSMIT_LUS (cluster \ dev --)
In COMM/3705 emulation, specifies the cluster controller and the logical unit device as the destination for the transmitted message.

Example:
Select cluster controller 4 and device 8 to transmit a message.

```
4 8 SET_TRANSMIT_LUS
```

SET_TRANSMIT_LU (LU --)
In CLST/3274 emulation, specifies the logical unit device from which to transmit a message to the host system.

Example:
Select logical unit device 23 on cluster controller 4 to transmit a message to the host system.

```
-1 23 4 ACT           ( Activate cluster controller 4 and device 23 )
23 SET_TRANSMIT_LU    ( Define logical unit device 23 to transmit )
```

NOTE
The specified logical unit device is attached to the last activated cluster controller. Although 32 CU's and the associated 32 LU's can be emulated simultaneously; only one CU and one LU can emulate the CLST/LU data transmission.

9.2 Timers and Counters

Timers and counters are used in BSC Emulation to simulate the appropriate responses and actions in accordance with the IBM 3270 Information Display System: 3274 Control Unit Description and Programmer's Guide, GA23-0061-2. publication for point-to-point (3780) and multipoint (3705/3274) protocol procedures. The following timers are used by the emulation:

Timer #	Constant
2	ACKNOWLEDGE_TO
3	RECEIVE_TO
5	POLL_TO
6	SELECT_TO
7	LEAD_TO

Table 9–2 Emulation Timers

NOTE
See Tables 11–1 and 11–2 for state descriptions.

READ–REQUEST–FLAG (-- address)
Contains a flag which causes the COMM/3705 emulation to enter the odd receive or odd transmission state after the first transmitted message.

ACKNOWLEDGE_TO (--2)

Acknowledgement (ENQ) timer. The timer duration is determined by the value stored in the ACKNOWLEDGE_TO_LENGTH variable.

In CLST/3274 emulation, this timer is started automatically when:

- a general POLL is received in the idle state and the cluster controller has transmitted its queued message;
- an ACK0 is received in the even read state and the cluster controller has transmitted its queued message;
- an ACK1 is received in the odd read state and the cluster controller has transmitted its queued message; or
- a timeout of the acknowledgement timer is received in the odd or even read state and the maximum number of ENQ retries, as stored in the ACKNOWLEDGE_TO_MAX variable, has not been reached.

In COMM/3705 emulation, this timer is started automatically when:

- an ACK0 is received in the select state, the communications controller transmits a message, and the READ-REQUEST-FLAG is not set;
- an ACK0 is received in the even write state and the communications controller transmitted its queued message;
- an ACK1 is received in the odd write state and the communications controller has transmitted its queued message; or
- a timeout of the acknowledgement timer is received in the odd or even write state and the maximum number of ENQ retries, as stored in the ACKNOWLEDGE_TO_MAX variable, has not been reached.

Example:
Start the acknowledge timer.
```
ACKNOWLEDGE_TO ACKNOWLEDGE_TO_LENGTH @ START_TIMER
```

ACKNOWLEDGE_TO_LENGTH (--address)

Contains the duration, in tenths of seconds, of the acknowledgement timer (default value is 3 seconds).

Example:
Set the acknowledge timer duration to 2 seconds.
```
20 ACKNOWLEDGE_TO_LENGTH !
```

ACKNOWLEDGE_TO_MAX (--address)

Contains the maximum number of acknowledgement retries when the emulation partner has not responded to a transmitted message (default is 3). When the acknowledgement timer expires and the emulation is in an appropriate state, an ENQ is transmitted when the current number of retries is less than the value stored in this variable.

When the maximum number of retries has been reached:

- the cluster controller returns to the idle state; or
- the communication controller polls or selects the next device.

Example:
Set the maximum number of ENQ retries to 1.
```
1 ACKNOWLEDGE_TO_MAX !
```

RECEIVE_TO (--3)

Receive timer. The timer duration is determined by the value stored in the RECEIVE_TO_LENGTH variable.

In CLST/3274 emulation, this timer is started automatically when:

- a SELECT is received in the idle state. The cluster controller transmits an ACK0;
- a TTD or an aborted frame is received in the odd or even write state. The cluster controller transmits a NAK; or
- a timeout of the receive timer is received in the odd or even write state and the maximum number of retries, as stored in the RECEIVE_TO_MAX variable, has not been reached.

In COMM/3705 emulation, this timer is started automatically when:

- a message ending with a ETB or ETX character is received in the general/specific poll state and no reverse interrupt condition has occurred. The communications controller transmits an ACK1 or an RVI if a reverse interrupt condition has occurred;
- an ACK0 is received in the select state, a queued message is transmitted, and the READ-REQUEST-FLAG is not true;
- a TTD or an aborted frame is received in the odd or even read state. A NAK is transmitted;
- a timeout of the receive timer is received in the odd or even read state and the maximum number of retries, as stored in the RECEIVE_TO_MAX variable, has not been reached; or
- a timeout of the receive timer is received in the RX/EOT_MP state and the maximum number of retries, as stored in the RECEIVE_TO_MAX variable, has not been reached.

Example:
Start the receive timer.
```
START_TO START_TO_LENGTH @ START_TIMER
```

RECEIVE_TO_LENGTH (--address)

Contains the duration, in tenths of seconds, of the receive timer (default is 1 second).

Example:
Set the receive timer duration to 2 seconds.
```
20 RECEIVE_TO_LENGTH !
```

RECEIVE_TO_MAX (--address)

Contains the maximum number of receive retries with no response from the emulation (default is 20). When the receive timer expires and the specified maximum number of receive retries is reached, CLST/3274 emulation returns to the idle state. When the maximum number is reached in COMM/3705 emulation, the next POLL or SELECT is transmitted.

Example:
Set the maximum number of receive retries to 1.
```
1 RECEIVE_TO_MAX !
```

POLL_TO (--5)
Poll timer. The timer duration is determined by the value stored in the POLL_TO_LENGTH variable.

In COMM/3705 emulation, this timer is started automatically when:

- a POLL is transmitted; or
- a timeout of the poll timer is received and the maximum number of poll retries, as stored in the POLL_TO_MAX variable, has not been reached.

Example:
Start the poll timer.
```
POLL_TO POLL_TO_LENGTH @ START_TIMER
```

POLL_TO_LENGTH (--address)
Contains the duration, in tenths of seconds, of the poll timer (default is 3 seconds).

Example:
Set the poll timer duration to 2 seconds.
```
20 POLL_TO_LENGTH !
```

POLL_TO_MAX (--address)
Contains the maximum number of poll retries when the emulation partner has not responded to a POLL (default is 3). When the poll timer expires and the number of retries is less than the value stored in this variable, the emulation re-polls the device. If the number of retries equals the value stored in POLL_TO_MAX, the communications controller polls or selects the next device.

Example:
Set the maximum number of poll retries to 1.
```
1 POLL_TO_MAX !
```

SELECT_TO (--6)
Select timer. The timer duration is determined by the value stored in the SELECT_TO_LENGTH variable.

In COMM/3705 emulation, this timer is started automatically when:

- a user transmit request is queued for the current device. The emulation transmits an EOT, a SELECT, starts the select timer, and goes to the select state;
- a WACK is received in the select state and the number of WACK's received is less than the value stored in the WACK_MAX variable. The communications controller transmits an ENQ; or
- a timeout of the select timer is received and the maximum number of ENQ retries as stored in the SELECT_TO_MAX variable has not been reached. The communications controller sends out an ENQ.

This timer stops when the emulation partner responds to a SELECT with an ACK0.

Example:
Start the select timer.
```
SELECT_TO SELECT_TO_LENGTH @ START_TIMER
```

SELECT_TO_LENGTH (--address)

Contains the duration, in tenths of seconds, of the select timer (default is 3 seconds).

Example:
Set the select timer duration to 2 seconds.

```
20 SELECT_TO_LENGTH !
```

SELECT_TO_MAX (--address)

Contains the maximum number of ENQ retries when the emulation partner has not responded to a SELECT (default is 3). When the select timer expires and the number of timeouts is less than the value stored in this variable, the communications controller transmits an ENQ and restarts the select timer. When the number of timeouts equals the value stored in SELECT_TO_MAX, the emulation polls or selects the next defined device.

Example:
Set the maximum number of ENQ retries to 1 when the select timer expires.

```
1 SELECT_TO_MAX !
```

WACK_MAX (--address)

Contains the maximum number of consecutive WACK responses from an emulation partner (default is 3). When a WACK response is received, the current WACK counter is incremented and then compared with this variable.

In CLST/3274 emulation in odd or even write state:

- when the number of received WACK's is less than the value in WACK_MAX, an ENQ is transmitted; or
- when the number of received WACK's is equal to the value in WACK_MAX, an EOT is transmitted and the emulation goes to the idle state.

In COMM/3705 emulation:

- in select state, when the number of received WACK's is less then the value in WACK_MAX, an ENQ is transmitted. When the maximum value is reached, an EOT is transmitted, and the emulation polls or selects the next device;
- in odd or even write, when the number of received WACK's is less than the value in WACK_MAX, an ENQ is transmitted. When the maximum value is reached, the emulation polls or selects the next device.

Example:
Set the maximum number of ENQ retries to 1 when a WACK is received following a SELECT.

```
1 WACK_MAX !
```

LEAD_TO (--7)
Lead transition timer. The timer duration is determined by the value stored in the LEAD_TO_LENGTH variable and is used when RTS/CTS control leads are enabled.

'TO DTE' Interface
When the RTS lead is on, control characters or message blocks are queued for transmission and the lead transition timer is started. When the RTS lead is off, the timer is stopped and the queued character or blocks are transmitted. If this timer expires, a notice is displayed advising that the RTS lead is on.

'TO DCE' Interface
When the CTS lead is off, control characters or message blocks are queued for transmission and the lead transition timer is started. When the CTS lead is on, the timer is stopped and the queued character or blocks are transmitted. If this timer expires, a notice is displayed advising that the CTS lead is off.

Example:
Start the lead transition timer.
```
LEAD_TO LEAD_TO_LENGTH @ START_TIMER
```

LEAD_TO_LENGTH (--address)
Contains the duration, in tenths of seconds, of the lead transition timer (default is 3 seconds).

Example:
Set the lead transition timer duration to 2 seconds.
```
20 LEAD_TO_LENGTH !
```

NAK_MAX (--address)
Contains the maximum allowed negative acknowledgements (default is 3). When a NAK is received, the current NAK counter is incremented and then compared with this variable.

In CLST/3274 emulation in odd or even read state:
- when the number of received NAK'S is less than the value in NAK_MAX, the current message is retransmitted; or
- when the number of received NAK'S is equal to the value in NAK_MAX, an EOT is transmitted and the emulation goes to the idle state.

In COMM/3705 emulation:
- in select state, when the number of received NAK's is less than the value in NAK_MAX, an ENQ is transmitted. When the maximum value is reached, an EOT is transmitted and the emulation polls or selects the next device; or
- in odd or even write state, when the number of received NAK's is less than the value in NAK_MAX, the current message block is retransmitted. When the maximum value is reached, an EOT is transmitted and the emulation polls or selects the next device.

STOP_TIMERS (--)
Stops the emulation timers until automatically restarted as described previously, or manually started with the START_TIMER command.

YES_TIMERS (--)
Returns emulation timeout durations to the default values.

10 EMULATION ARCHITECTURE

This section describes the structure of the BSC Emulation. The IDACOM BSC Emulation program is a combination of the complete BSC Monitor application package together with an emulation state machine. All commands available in the BSC Monitor are also available in the BSC Emulation. The program's data flow is detailed for the reception of protocol events (messages, lead changes, or timers) and generating responses to these events.

Received messages are first decoded and then passed on to the test manager if a test script is running. The test manager can generate data, start timers, etc., in response to the received message, or strictly recognize that a particular message has been received with no associated output. After the test manager has processed the received message, the BSC state machine processes the received message and generates any necessary protocol responses.

10.1 Live Data

The BSC Monitor decodes any received/transmitted messages and displays them for user interpretation, while the BSC state machine interprets the received messages and forces some action (usually transmitting a message, EOT, NAK, etc.).

The emulation receives events from the interface and processes them as shown in Figure 10-1.

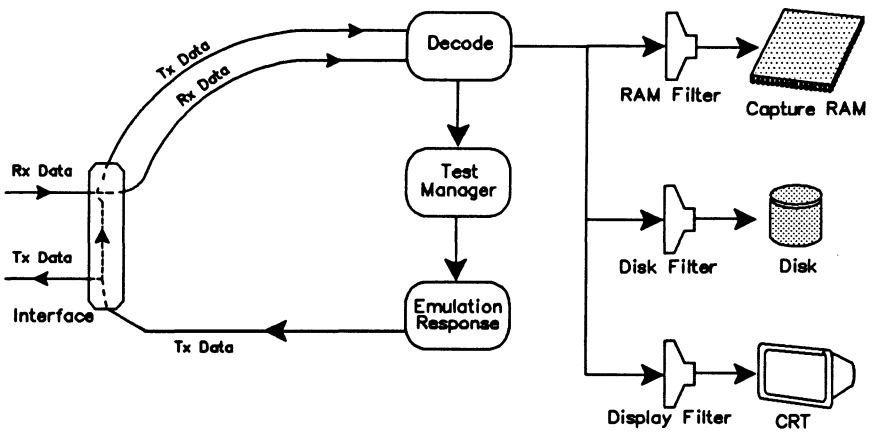


Figure 10–1 BSC Emulation Data Flow Diagram – Live Data

By default, the BSC Emulation captures the received/transmitted data in the capture RAM buffer and displays it on the screen in a short format report. The BSC Emulation is running (active) and responds to all data link control sequences. The emulation can be enabled or disabled using the EMUL_ON or EMUL_OFF commands.

Display topic
Live Data function key

MONITOR (--)
Selects the live data mode of operation. All incoming events and transmitted messages are decoded and displayed in real-time.

EMUL_ON (--)
Enables the automatic emulation; the BSC state machine responds to all incoming messages according to the BSC protocol (default).

Emulation topic
Run Emulation function key (highlighted)

EMUL_OFF (--)
Disables the automatic emulation; the BSC state machine does not transmit any messages without manual or test program intervention.

Emulation topic
Run Emulation function key (not highlighted)

10.2 Playback

Data can be played back from either capture RAM or disk without interfering with an active test (i.e. dropping the link) as shown in Figure 10-2.

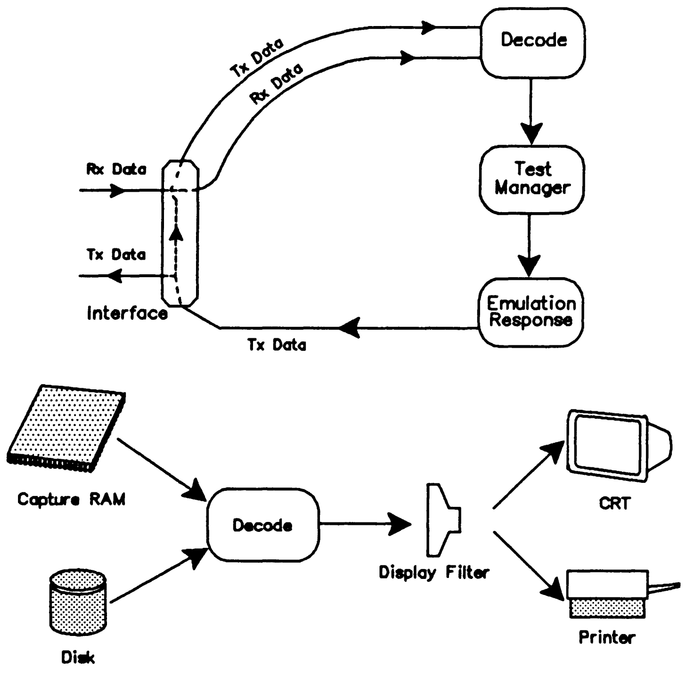


Figure 10–2 BSC Emulation Data Flow Diagram – Offline Processing

FROM_CAPT HALT
Display topic
Playback RAM function key

FROM_DISK HALT PLAYBACK
Display topic
Playback Disk function key

HALT (--)
Selects the playback mode of operation. Data is retrieved from capture RAM or disk, decoded, and displayed or printed. Capture RAM is suspended in this mode.

10.3 Simultaneous Live Data and Playback

Live data can be recorded to disk while playing back data from capture RAM.

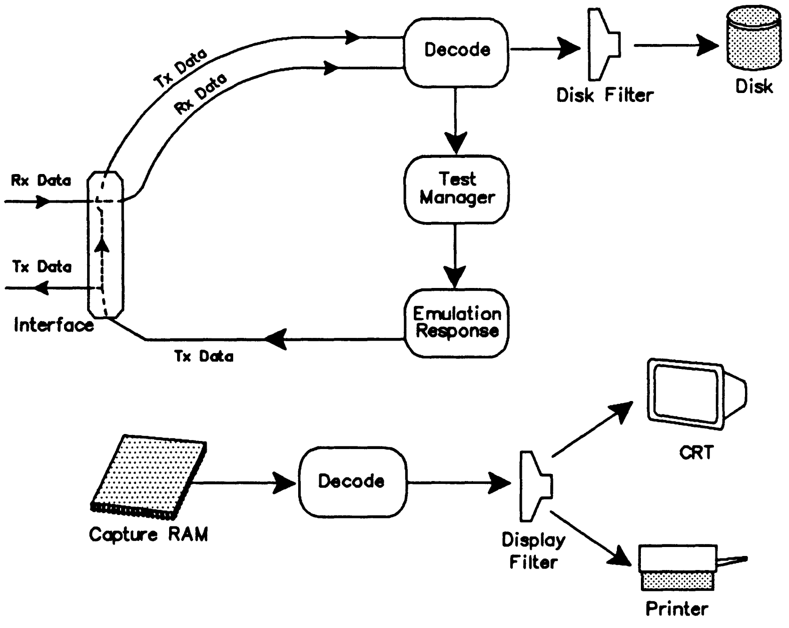


Figure 10–3 BSC Emulation Data Flow Diagram – Freeze Mode

FROM_CAPT FREEZE
Capture topic
Record to Disk function key
Display topic
Playback RAM function key

FREEZE (--)
Enables data to be recorded to disk while data from capture RAM is played back.

11 EMULATION RESPONSE

The BSC Emulation is implemented as a single layer, state driven protocol emulation for 3705/3274 (multipoint) protocol procedure. There are separate program modules set for each emulation mode (COMM/3705 or CLST/3274). These modules communicate with each other to implement protocol response behavior.

The emulation has been set up to run as:

- an automatic simulation which operates precisely in accordance with IBM 3270 Information Display System: 3274 Control Unit Description and Programmer's Guide, GA23-0061-2 publication for point-to-point (3780) and multipoint (3705/3274) protocol procedures;
- a semi-automatic tester. The test manager is used to build and execute test scenarios to test responses and for generation of errors (see Section 12); and
- a manual tester. The test is controlled from the user's keyboard.

11.1 Emulation State Machines

To ensure correct protocol operation, state machines have been implemented. Based on input events (i.e. received messages) transitions from one state to another are made in accordance with the IBM 3270 Information Display System: 3274 Control Unit Description and Programmer's Guide, GA23-0061-2 publication.

BSC_DISPLAY_STATE (--)
Displays the current state of the emulation in the Notice Window.

BSC_CHANGE_STATE (state number--)
Changes the state machine to the specified state. Valid values are listed in Tables 11-1 and 11-2.

Example:
Set the emulation to the general/specific poll state.

```
7 BSC_CHANGE_STATE
```

State #	Condition	Constant	Expected Response
7	General/Specific Poll State	(POLL_MP)	Receive Timeout, EOT, ETX, ETB
8	Select State	(TX/SELECT_MP)	ACK0, WACK, NAK
9	Odd Write State	(TX/ODD_MP)	ACK1, NAK, WACK
10	Even Write State	(TX/EVEN_MP)	ACK0, NAK, WACK
11	Odd Read State	(RX/ODD_MP)	EOT, ETB, ETX, ENQ, TTD, ABORT, ILLEGAL
12	Even Read State	(RX/EVEN_MP)	EOT, ETB, ETX, ENQ, TTD, ABORT, ILLEGAL
13	Reverse Interrupt Condition	(RX/EOT_MP)	EOT, ENQ, Receive Timeout

Table 11-1 COMM/3705 States

State #	Condition	Constant	Expected Response
14	Idle State	(IDLE_CC)	General/Specific Poll, Select Sequence
15	Odd Read State	(TX/ODD_CC)	ACK1, RV1, NAK, WACK
16	Even Read State	(TX/EVEN_CC)	ACK0, RV1, NAK, WACK
17	Odd Write State	(RX/ODD_CC)	EOT, ETB, ETX, ENQ, TTD (Aborted), BCC Error (Illegal)
18	Even Write State	(RX/EVEN_CC)	EOT, ETB, ETX, ENQ, TTD (Aborted), BCC Error (Illegal)

Table 11-2 CLST/3274 States

11.2 Automatic Responses

The state machines normally handle the protocol automatically, i.e. automatic responses to received messages.

EMUL_ON (--)
Enables the automatic emulation; the BSC state machine responds to all incoming messages according to the BSC protocol (default).

EMUL_OFF (--)
Disables the automatic emulation; the BSC state machine does not transmit any messages without manual or test program intervention.

Protocol state change reports can be displayed, captured, or recorded to disk along with Bisync data. These change reports are useful for tracing protocol or test manager operation.

STATE-TRACE (-- address)
Contains a flag which enables (1, default) or disables (0) a trace report line for every protocol state change.

11.3 Send Commands

The following variables and commands are used to control the transmission of poll sequences, select sequences, control characters, and data messages. These variables and commands are all used in both emulation modes, except poll and select sequence commands which are only used in COMM/3705 emulation.

Poll Sequences

GENERAL (--)
Specifies that subsequent transmitted polls are general poll sequences.

SPECIFIC (--)
Specifies that subsequent transmitted polls are specific poll sequences.

POLL_ON (--)
Starts continuous transmission of a general/specific poll sequence by COMM/3705 if no user data is pending for transmission. If user data is pending, a select sequence will be sent and, after receiving a positive acknowledgement from the polled unit, a data message will be transmitted.

Send topic
Start Polling function key

POLL_OFF (--)
Stops transmission of general/specific poll sequences.

Send topic
Stop Polling function key

CURRENT_POLL_DEV (-- address)
Contains the polled logical unit device identifier. Valid values are 0 through 31.

CURRENT_POLL_LU (-- address)
Contains the polled cluster controller identifier. Valid values are 0 through 31.

POLL_CURR (--)
Transmits a general/specific poll sequence to the *current* cluster controller or logical unit device.

NOTE
The current cluster controller and logical unit device are contained in the CURRENT_POLL_DEV and CURRENT_POLL_LU variables respectively.

POLL_NEXT (--)
Transmits a general/specific poll sequence to the *next* cluster controller or logical unit device.

NOTE
When only one device is defined, next and current polls are the same.

Select Sequences

MESSAGE–SEND (-- address)
Contains a flag which enables (1) or disables (0) the transmission of a user–defined message after receiving a positive acknowledgement to a select sequence.

SEL_CURR (--)
Transmits a select sequence to the *current* cluster controller or logical unit device (according to the poll train array). When MESSAGE-SEND is set to 1, the COMM/3705 emulation transmits a data message after receiving a positive acknowledgement to a select sequence. When MESSAGE-SEND is set to 0, no user data is sent after receiving a positive acknowledgement to a select sequence.

SEL_NEXT (--)
Transmits a select sequence to the *next* cluster controller or logical unit device (according to the poll train array). When MESSAGE_SEND is set to 1, the COMM/3705 emulation transmits user data after receiving a positive acknowledgement to a select sequence. When MESSAGE_SEND is set to 0, no data message is sent after receiving a positive acknowledgement.

NOTE
Refer to the SET_MESSAGE command to define a data message.

NOTE
When only one device is defined, next and current SELECT sequences are the same.

SELECT_TX (--)
Transmits a select sequence to the *next* cluster controller or logical unit device. No data messages are transmitted.

SELECT function key

NOTE
A data message can be transmitted if the AUTO_TRANSMIT_ON command has been called prior to SELECT_TX.

Control Characters

EOT_TX (--)
Transmits EOT control characters.

Send topic
EOT function key

ACK0_TX (--)
Transmits ACK0 control characters.

Send topic
ACK0 function key

ACK1_TX (--)
Transmits ACK1 control characters.

Send topic
ACK1 function key

NAK_TX (--)
Transmits NAK control characters.

NAK function key

WACK_TX (--)
Transmits WACK control characters.

Send topic
WACK function key

ENQ_TX (--)
Transmits ENQ control characters.

Send topic
ENQ function key

RVI_TX (--)
Transmits RVI control characters.

Send topic
RVI function key

RVI_REQUEST (--)
The communications controller transmits an RVI sequence on the next received data message ending with an ETX character.

Data Messages

SET_MESSAGE (string--)

Defines a data message in the current message buffer. Maximum message length is 60 characters.

Example:
Define the following message.

```
" IDACOM brings you the PT Protocol Tester" SET_MESSAGE
```

Send topic
Transmit Message function key
New Message function key

NOTE
Message length can be extended using the ADD_TEST command.

ADD_TEXT (string-- overflow count)

Appends additional data to the current message buffer, previously defined using the SET_MESSAGE command.

Example:
Add the following additional text to the current message buffer.

```
" IDACOM leads you into the future with the PT500" ADD_TEXT
```

Send topic
Transmit Message function key
Append Message function key

CLEAR_MESSAGE (--)

Empties the data message buffer.

WARNING
Do not clear the data message buffer while sending a message.

SEND (--)

Unconditionally transmits the current user-defined data message.

NOTE
Data messages are automatically enclosed in STX/ETX control characters and appended with CRC bytes prior to transmission.

AUTO_TRANSMIT_ON (--)

In COMM/3705 emulation, automatically transmits the *current* data message after receiving a positive acknowledgement to a select sequence.

In CLST/3274 emulation, automatically transmits the *current* data message when polled.

Send topic
Auto Transmit function key (highlighted)

NOTE
The transmitting logical unit device and cluster controller must be previously defined using the SET_TRANSMIT_LU command.

AUTO_TRANSMIT_OFF (--)

Disables autotransmission (default).

Send topic
Auto Transmit function key (not highlighted)

MAXIMUM_DATA_BLOCK_SIZE (--address)

Contains the maximum transmission message block size (default is 255; maximum is 5000).

Example:
Set the maximum transmission message block size to 3000 characters.

```
3000 MAXIMUM_DATA_BLOCK_SIZE !
```

The following variables are used when transmitting message blocks and contain the address and length of the first, next, or current message block.

FIRST_BLOCK (--address\length)
Contains the address and length of the first transmission message block buffer, or returns 0 if the buffer is empty.

Example:
Send the first message block of a message if an ACK0 is received and the buffer is not empty, then go to state 4, otherwise enter into state 7.

```
3 STATE{
     ACK0 1 ?RX              ( Detecting a ACK0 character )
     ACTION{
         FIRST_BLOCK         ( Retrieve 1st message block buffer addr & length )
         ?DUP                ( Is the message block buffer not empty? )
         IF                  ( YES:  send message block )
             BSC_TX          ( Send message block )
             4 NEW_STATE     ( Go State 4 )
         ENDIF               ( NO:  go to state 7 )
             7 NEW_STATE
     }ACTION
  }STATE
```

NOTE
The first transmitted message block of a message must be transmitted using FIRST_BLOCK. Succeeding message blocks must be transmitted using the NEXT_BLOCK or CURRENT_BLOCK command.

CURRENT_BLOCK (-- address/length | 0)

Contains the address and length of the transmitted message block buffer, or returns 0 if the buffer is empty.

Example:
Retransmit the current message block after receiving a NAK. If the buffer is empty, send an EOT control character and enter state 5.

```
3 STATE{
     NAK 1 ?RX                  ( Detecting a NAK acknowledgement )
     ACTION{
          CURRENT_BLOCK         ( Get address and size of buffer to retransmit )
          ?DUP                  ( Is message block buffer not empty? )
          IF                    ( YES:  retransmit last message block )
               BSC_TX           ( Retransmit message block )
               STOP_TIMERS      ( NO:  stop timers )
               EOT_TX           ( Transmit an EOT character )
               5 NEW_STATE      ( Enter state 5 )
          ENDIF
     }ACTION
}STATE
```

NOTE
The first transmitted message block of a message must be transmitted using FIRST_BLOCK. Succeeding message blocks must be transmitted using the NEXT_BLOCK or CURRENT_BLOCK command.

NEXT_BLOCK (-- address/length | 0)

Contains the address and length of the *next* transmission message block buffer, or returns 0 if the buffer is empty.

Example:
Transmit the next message block and clear counters, after receiving a positive acknowledgement to the first transmitted message block. If the buffer is empty, send an EOT control character and stop timers.

```
4 STATE{
      ACK1 1 ?RX                           ( Detecting an ACK1 acknowledgement )
      ACTION{
          NEXT_BLOCK                       ( Get next message block to be )
                                           ( transmitted )
          ?DUP                             ( Is there more data to send? )
          IF                               ( YES:  more data left to send )
              0 INCORRECT_ACK_COUNT !      ( Clear counters )
              0 ACKNOWLEDGE_TO_COUNT !
              0 WACK_COUNT !
              0 NAK_COUNT !
              BSC_TX                       ( Transmit the last message block )
          ELSE                             ( NO:  no more data left to send )
              EOT_TX                       ( Transmit EOT character )
              STOP_TIMERS
          ENDIF
      }ACTION
  }STATE
```

STATUS_BLOCK (--address\length)

Contains the address and length of the transmission message block buffer without STX and ETX control characters. STATUS_BLOCK can be used to format a customized message block such as a status message for transmission.

Example:
After receiving a specific poll, transmit a status message with the status bit DC (data check), US (unit specify), and DE (device equipment) set, then enter state 4.

```
2 STATE{
      SPECIFIC_POLL 1 ?RX
      ACTION{
          X" 016CD902C5C4C4C503"       ( Create status message with the appropriate )
                                       ( header, addresses, and control characters: )
                                       ( SOH, %, R, STX, 3274 POLL addr, Dev addr, )
                                       ( status byte 0 and 1 "DC, US, DE", and ETX )

          SET_MESSAGE                  ( Set string into transmission buffer )
          STATUS_BLOCK                 ( Select format of transmission message )
                                       ( block )
          BSC_TX                       ( Send the status message block )
          4 NEW_STATE                  ( Enter state 4 )
      }ACTION
  }STATE
```

BSC_TX (address\length --)
Transmits the transmission message block.

11.4 Error Generation

The following commands toggle the error bit contained in the ERROR-GEN variable to generate errors in transmitted messages.

ERROR-GEN (-- address)
Contains the bit setting to create invalid transmit messages (i.e. ENQ, STX, ETX, and/or BCC errors).

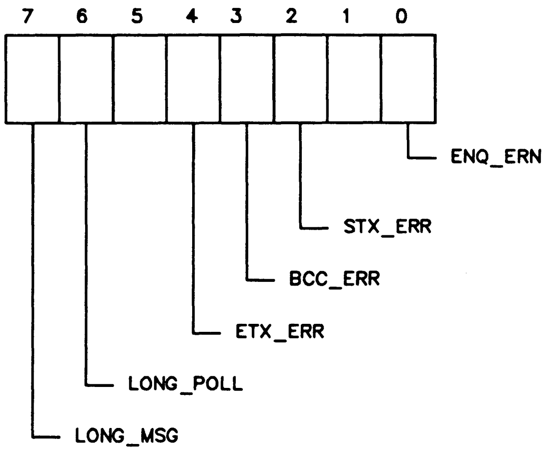


Figure 11-1 ERROR-GEN Bit Position

ENQ_ERR (--)
Toggles the ENQ error bit. When set to 1, all poll or select sequences are transmitted without an ENQ character.

Example:
Set the ENQ error bit to 1 to force poll and select errors.

```
ERROR-GEN @ 1 AND 0=          ( Is bit set? )
IF                            ( No )
  ENQ_ERR                     ( Set it )
ENDIF
```

STX_ERR (--)

Toggles the STX error bit. When set to 1, message blocks are transmitted with an invalid STX character (i.e. hex 0D instead of hex 02).

Example:
Set the STX error bit so that message blocks are transmitted with an invalid STX character.

```
ERROR-GEN @ 4 AND 0=          ( Is bit set? )
IF                            ( No )
  STX_ERR                     ( Set it )
ENDIF
```

ETX_ERR (--)

Toggles the ETX error bit. When set to 1, message blocks are transmitted with an invalid ETX character (i.e. hex 0D instead of hex 03).

Example:
Set the ETX error bit so that message blocks are transmitted with an invalid ETX character.

```
ERROR-GEN @ 16 AND 0=         ( Is bit set? )
IF                            ( No )
    ETX_ERR                   ( Set it )
ENDIF
```

BCC_ERR (--)

Toggles the BCC error bit. When set to 1, message blocks are sent with an invalid CRC error.

Example:
Set the BCC error bit so that message blocks are transmitted with a CRC error.

```
ERROR-GEN @ 8 AND 0=          ( Is bit set? )
IF                            ( No )
    BCC_ERR                   ( Set it )
ENDIF
```

LONG_POLL (--)

Toggles the long poll error bit. When set to 1, subsequent polls (general, specific, or select) are sent with the header address repeated twice.

Example:
Set the long poll error bit so that long polls are transmitted with the header address repeated twice.

```
ERROR-GEN @ 64 AND 0=         ( Is bit set? )
IF                            ( No )
    LONG_POLL                 ( Set it )
ENDIF
```

LONG_MSG (--)
Toggles the long message bit. When set to 1, the largest permitted transmitted message block size is the value indicated in the MAXIMUM_DATA_BLOCK_SIZE variable. Otherwise, the largest permitted transmission message block size is 255 (default) characters.

GOOD (--)
Resets the STX, ETX, and BCC error bits. When set to 1, subsequent message blocks are sent with valid control characters and CRC bytes.

12 TEST MANAGER

IDACOM has developed a comprehensive set of tools for the development of test scripts. These test scripts, written using the ITL language, control the operation of the BSC Monitor/Emulation application.

For a complete explanation of the test manager and tools available, see the Programmer's Reference Manual.

This section reviews basic ITL components and describes the protocol event and action commands specific to BSC.

12.1 ITL Constructs

Following is a brief description of test manager constructs. For more details and examples, refer to the Programmer's Reference Manual.

TCLR (--)
Initializes the test manager. Any existing test suites already in memory are cleared. The current state is set to 0. All test scenarios should start with the TCLR command.

STATE_INIT{ }STATE_INIT (n--)
Brackets the execution sequence performed prior to entering a state. The initialization logic for a state is executed independently of how it was called.

This initialization procedure can be used for any state but is not compulsory. STATE_INIT{ must be preceded by the number of the state being initialized, eg. 0 STATE_INIT{.

The STATE_INIT{ }STATE_INIT clause is executed only once each time the state is entered from another state.

STATE{ }STATE (n--)
Brackets a state definition. STATE{ must be preceded by the number of the state. Valid values are 0 through 255. State 0 must be defined within an ITL program. If not, the test manager will not run the script. If multiple states are defined with the same number in the test script, the test manager uses the latest definition.

ACTION { }ACTION (f --)
Brackets the set of tasks, decisions, and outputs which execute once the expected event is received by the test manager. There must be at least one action defined for each expected event. The action is executed when the flag is true (non-zero).

NEW_STATE (n--)
Executes the initialization logic of the specified state (providing STAT_INIT{ }STAT_INIT is defined) and establishes the state to be executed for the next event. Any remaining action code for the current state is then executed. It must be preceded with a valid state number and be inside the ACTION{ }ACTION brackets. This command is not mandatory if no state change is desired.

TM_STOP (--)
Stops the execution of the test script. The test suite remains in memory and can be re-executed until another test script is loaded.

SEQ{ }SEQ (number--)
Brackets a definition of tasks and outputs which execute as part of the state machine action. SEQ{ expects a single integer which is the sequence number. Up to 256 sequences are supported. Valid values are 0 through 255. The SEQ{ }SEQ partners are extremely useful when more than one action sequence calls the same tasks and outputs. The SEQ{ }SEQ definition is defined outside the ACTION{ }ACTION definition and then called by the RUN_SEQ command.

This is an alternate mechanism to generate colon definitions. This mechanism causes the equivalent of a colon definition (now accessed via a numeric identifier) to be compiled into the test script dictionary rather than the user dictionary. Refer to the Programmer's Reference Manual.

RUN_SEQ (number--)
Executes a specified set of tasks defined in a SEQ{ }SEQ definition. It is called inside an ACTION{ }ACTION definition and must be preceded with a defined sequence number.

LOAD_RETURN_STATE (number--)
Permits the test script writer to program the equivalent of subroutine calls (used with RETURN_STATE). LOAD_RETURN_STATE sets the state to which control is to be returned. LOAD_RETURN_STATE must be within the action field; nesting is not permitted.

RETURN_STATE (--)
Returns control to the state specified by LOAD_RETURN_STATE from a state subroutine call.

NEW_TM (filename--)
Loads and compiles the specified file and then starts the test manager at state 0. It can be included as part of the action field to load and execute another scenario.

12.2 Event Recognition

During test script execution, any event received by the test manager is evaluated to determine if it matches the event-specifier of the first action within that state. If the evaluation does not return a true value, the following action clauses are evaluated in a sequential manner. Once an event evaluates true, the subsequent action clauses in that particular state are not examined.

Physical Layer

See the Programmer's Reference Manual for a description of layer 1 events (i.e. control lead transitions) when the application is running on a WAN interface.

Received Messages

ITL provides recognition of protocol specific messages, anchored or unanchored comparison of user-defined octets, CRC errors, and illegal and aborted messages.

Messages are decoded and stored in various communication variables when received by the monitor or emulation. This decoded information is used by the test manager to identify a particular event and can be obtained by using the @ fetch command.

FRAME-ID (--address)
Contains an identifier which indicates the type of data link control sequence received. This variable is set when a control sequence or a single control character is initiated between the cluster controller and the host system. Valid identifiers are listed in Table 12-1.

Request/Response (RU) Identifier	Description
GENERAL_POLL	TCU general poll sequence
SPECIFIC_POLL	TCU specific poll sequence
SELECT	TCU select sequence (request to write)
ACK0	Even acknowledgements
ACK1	Odd acknowledgements
NAK	Negative acknowledgements
WACK	Wait for acknowledgement
EOT	End of transmission response
ENQ	Enquiry
TTD	Forward abort sequence (STX ENQ)
BCC_ERROR	Incorrect BCC sequence
RVI	Reverse interrupt response
ETX_DATA	End of text indicates last transmission block
ETB_DATA	End of block indicates end of transmission block
ILLEGAL	Message does not conform to BSC protocol
SHORT_FRAME	Short read message is received
ABORTED	Aborted message is received

Table 12-1 Data Link Control Sequence Identifiers

PACKET–ID (--address)

Contains an identifier which indicates the type of message (command or attention) last received. This variable is set when one of these messages is initiated between the host system and device. When PACKET–ID is set to 0, no commands or attention identifiers are transmitted. If a message other than a command or attention identifier is received, PACKET–ID will contain 0.

Valid identifiers for received command messages are listed in Table 12–2.

Command Identifier	Description
READ_MODIFIED	Read modified command message
READ_MODIFIED_ALL	Read modified all command message
READ_BUFFER	Read buffer command message
COPY_COMMAND	Copy command message
WRITE	Write command message
ERASE/WRITE	Erase/write command message
ERASE/WRITE_ALT	Erase/write alternate command message
ERASE_ALL_UNPROT	Erase all unprotected command message
WRITE_STRUCT_FIELD	Write structured fields command message

Table 12–2 Command Identifiers

NOTE
Write structured fields are not supported in an ASCII environment, since the use of structured fields requires the full 8 bits of a byte.

Valid identifiers for received attention identifier messages are listed in Table 12-3.

Attention Identifier (AID)	Description
PA1	Program attention key 1
PA2	Program attention key 2
PA3	Program attention key 3
ENTER	Enter key
PEN	Selector pen
TEST_REQ	Test requested
CLEAR_PART	Clear partition
STRUCTURED_FIELD	Structured field unbounded data
MAG_READER	Magnetic stripe reader
ID_READER	Operator ID reader AID
NO_AID_PRINTER	Printer unsolicited read, error, or unusual condition
NO_AID	Unsolicited read, error, or unusual condition
F_STATUS	Status message
CLEAR	Clear display completely
CLEAR_PART	Clear display partition
PF1 to PF24 keys	Program function keys
TRIGGER_ACTION	Trigger action
NO_MESSAGE	Chained message
ILLEGAL_WRITE	WRITE message does not confirm to BSC protocol
ILLEGAL_READ	READ message does not confirm to BSC protocol

Table 12–3 Attention Identifiers

NetWorkType (-- address)

Contains the network type identifier indicating the protocol procedure currently being monitored (i.e point-to-point, multipoint). This variable is updated every time a communication data link is successfully established. Valid identifiers are listed in Table 12-4.

Network Type	Description
BOTH * (--n)	Both multipoint and point to point procedure
MULTI*POINT (--n)	Multipoint (COMM/3705 & CLST/3274) procedure
POINT*POINT (--n)	Point-to-point 3780 procedure

Table 12–4 Network Type Identifiers

?RX (frame id#1 \ frame id#2 \...\ frame id#n \ n--flag)
Returns true if one of the specified data link control sequences is received. See Table 12-1 for valid identifiers.

Example:
Look for reception of a general or specific poll sequence. The user receives an audible alarm and a notice.

```
3 STATE
      GENERAL_POLL SPECIFIC_POLL 2 ?RX
      ACTION{
          BEEP
          " General or Specific poll sequence received" W.NOTICE
      }ACTION
}STATE
```

?RX_DTE (frame id#1 \ frame id#2 \...\ frame id#n \ n--flag)
Returns true if one of the specified data link control sequences is received from the DCE. See Table 12-1 for valid identifiers.

Example 1:
Look for reception of a select sequence using the monitor. The user receives an audible alarm and a notice.

```
3 STATE{
      SELECT 1 ?RX_DTE
      ACTION{
          BEEP
          " Select sequence received" W.NOTICE
      }ACTION
}STATE
```

Example 2:
Look for reception of a forward abort sequence using the emulation. The user receives an audible alarm and proceeds to state 4.

```
1 STATE{
      TTD 1 ?RX
      ACTION{
          BEEP
          4 NEW_STATE           ( Go to state 4 )
      }ACTION
}STATE
```

?RX_DCE (frame id#1 \ frame id#2 \...\ frame id#n \ n--flag)
Returns true if one of the specified data link control sequences is received from the DTE. See Table 12-1 for valid identifiers.

?RX_LU (LU#--flag)
Returns true if all messages transmitted from the host system are received by the specified logical unit device. Valid values are 0 through 31.

?RX_CU (CU#--flag)
Returns true if all messages transmitted from the host system are received by the specified cluster controller. Valid values are 0 through 31.

?MESS (packet id#1 \ packet id#2 \...\ packet id#n \ n--flag)
Returns true if one of the specified command or attention identifier messages, ending with an ETX or ETB character, is received. See Tables 12-2 and 12-3 for valid identifiers.

Example:
Look for reception of the PF1 through PF4 keys, increment PF_COUNT by 1, and go to state 3.

```
10 STATE{
      PF1 PF2 PF3 PF4 4 ?MESS  ( Detects specified PF keys )
      ACTION{
          1 PF_COUNT +!        ( Increment counter by 1 )
          3 NEW_STATE          ( Go to State 3 )
      }ACTION
}STATE
```

?MESS_DTE (packet id#1 \ packet id#2 \...\ packet id#n \ n--flag)
Returns true if one of the specified command or attention identifier messages is received from the DCE. See Tables 12-2 and 12-3 for valid identifiers.

?MESS_DCE (packet id#1 \ packet id#2 \...\ packet id#n \ n--flag)
Returns true if one of the specified command or attention identifier messages is received from the DTE. See Tables 12-2 and 12-3 for valid identifiers.

?DATA (string--flag)
Returns true if a user-defined character string is found in the data field of the received messages specified with the ?MESS commands.

This is an *anchored* match, i.e. a byte-for-byte match starting at the first byte of the data field. Any blocks with an ETX or ETB character are decoded and the data field type is identified.

NOTE
To accommodate "don't care" character positions, the question mark character for ASCII or hex 3F character can be used. L4-POINTER indicates the location of the first byte in the data field. See Appendix C for the positioning of L4_POINTER for different message types. The maximum string length is 80 characters. The received data field can be longer than the specified string.

Example:
Upon reception of the PF1 through PF4 keys from the DCE, search for the hex string X " 70115B". Save the received byte length in the BLOCK-LENGTH variable, receive an audible alarm, display a notice, and run sequence 1.

```
2 STATE{
      PF1 PF2 PF3 PF4 4 ?MESS_DTE X" 70115B" ?DATA AND
      ACTION{
          REC-LENGTH @
          BLOCK-LENGTH !
          BEEP
          " Hex pattern in data field received" W.NOTICE
          1 RUN_SEQ
      }ACTION
  }STATE
```

?DATA_DTE (string--flag)
Returns true if a user-defined character string is found in the data field of a message received from the DCE.

?DATA_DCE (string--flag)
Returns true if a user-defined character string is found in the data field of a message received from the DTE.

?RECV'D (string--flag)
Returns true if a user-defined character string is found in a received message.

This is an *anchored* match, i.e. a byte-for-byte match starting at the first byte of the received message block.

Example:
Search for the character string 'IDACOM Protocol Tester' in the received message.

```
" ?IDACOM Protocol Test" ?RECV'D
```

?RECV'D_DTE (string--flag)
Returns true if a user-defined character string is found in a message received from the DCE (anchored match - see ?RECV'D).

?RECV'D_DCE (string--flag)
Returns true if a user-defined character string is found in a message received from the DTE (anchored match - see ?RECV'D).

?SEARCH (string--flag)
Returns true if a user-defined character string is found in a received message.

This is an *unanchored* match, i.e. searches for an exact match anywhere in the input string.

Example:
Search for the character string 'IDACOM' in the received message.

```
" IDACOM" ?SEARCH
```

NOTE
To accommodate "don't care" character positions, the question mark character for ASCII, or a hex 3F character can be used. The maximum string length is 80 characters.

?SEARCH_DCE (string--flag)
Returns true if a user-defined character string is found in a message received from the DTE (unanchored match - see ?SEARCH).

?SEARCH_DTE (string--flag)
Returns true if a user-defined character string is found in a message received from the DCE (unanchored match - see ?SEARCH).

?ABORT (--flag)
Returns true if an aborted message is received.

?ABORT_DTE (--flag)
Returns true if an aborted message is received from the DCE.

?ABORT_DCE (--flag)
Returns true if an aborted message is received from the DTE.

Example:
Detect an aborted message received from the DTE.

```
2 STATE{
      ?ABORT_DCE
      ACTION{
          " Aborted message received from DCE side"  W.NOTICE
      }ACTION
  }STATE
```

?CRC_ERROR (--flag)
Returns true if a message with a CRC error is received.

?CRC_ERROR_DTE (--flag)
Returns true if a message with a CRC error is received from the DCE.

?CRC_ERROR_DCE (--flag)
Returns true if a message with a CRC error is received from the DTE.

Example:
Detect a message with a CRC error received from the DTE.

```
1 STATE{
        ?CRC_ERROR_DCE
        ACTION{
               " Invalid CRC value from DTE side was detected" W.NOTICE
        }ACTION
  }STATE
```

Timeout Detection

Timers 8 through 25 and 41 through 128 can be used in the test manager. Timer 34 is the test manager wakeup timer. The remaining timers are used within the application and should not be started or stopped in a test script.

TIMEOUT (--flag)
Returns true if any timer has expired.

Example:
In State 8, look for the expiration of any timer. The action is to display a trace statement.

```
8 STATE{
      TIMEOUT      ( Check for timeout of any timer )
      ACTION{
          T." A Timer has expired." TCR
       }ACTION
  }STATE
```

?TIMER (timer#--flag)
Returns true if the specified timer has expired.

Example:
In State 8, look for the expiration of timer 10. The action is to display a trace statement.

```
8 STATE{
     10 ?TIMER     ( Check for timeout of timer 10 )
     ACTION{
         T." Timer 10 has expired." TCR
      }ACTION
  }STATE
```

?WAKEUP (--flag)
Returns true if the wakeup timer has expired. The wakeup timer can be used to initiate action sequences immediately upon the test manager starting. Timer 34 is started for 100 milliseconds when the test manager is started after a WAKEUP_ON command has been issued. The default is WAKEUP_OFF.

Example:
In State 0, look for the expiration of the wakeup timer. The action is to prompt the user to press a function key, and then the test manager goes to State 1.

```
0 STATE{
       ?WAKEUP                  ( Check for timeout of wakeup timer )
       ACTION{
            T." To restart the test, press UF1." TCR
            1 NEW_STATE
        }ACTION
 }STATE
```

TIMER-NUMBER (--address)
Contains the number of the expired timer. Valid values are 1 through 128.

Function Key Detection

Refer to the Programmer's Reference Manual.

Interprocessor Mail Events

Refer to the Programmer's Reference Manual.

Wildcard Events

The BSC application supports the OTHER_EVENT command and the EVENT-TYPE variable. Refer to the Programmer's Reference Manual.

The EVENT-TYPE variable contains any one of the following constants: FRAME, LEAD*CHANGE, TIME*OUT, FUNCTION*KEY, or COMMAND_IND.

FRAME (--value)
A constant value in the EVENT-TYPE variable when the received event is a frame. The actual protocol type is in either the FRAME-ID or the PACKET-ID variables (see Tables 12-1, 12-2, and 12-3).

LEAD*CHANGE (--value)
A constant value in the EVENT-TYPE variable when the received event is a control lead transition. The actual lead transition is in the LEAD-NUMBER variable.

TIME*OUT (--value)
A constant value in the EVENT-TYPE variable when the received event is a timeout. The actual timer is in the TIMER-NUMBER variable.

FUNCTION*KEY (--value)
A constant value in the EVENT-TYPE variable when a function key or cursor key is detected. The actual key value is in the KEY-NUMBER variable.

NOTE
To detect function keys, it is advisable to use the ?KEY command. Refer to the Programmer's Reference Manual.

COMMAND_IND (--value)
A constant value in the EVENT-TYPE variable when an interprocessor mail indication is received. Refer to the Programmer's Reference Manual.

12.3 BSC Actions

All of the general actions explained in the Programmer's Reference Manual are supported in the BSC Monitor and Emulation.

Physical Layer Actions

The following emulation commands turn control leads on and off.

V.28/RS-232C Interface		
OFF to ON	ON to OFF	Description
RTS_ON	RTS_OFF	Request to send
CTS_ON	CTS_OFF	Clear to send
DSR_ON	DSR_OFF	Data set ready
CD_ON	CD_OFF	Carrier detect
DTR_ON	DTR_OFF	Data terminal ready
SQ_ON	SQ_OFF	Signal quality
RI_ON	RI_OFF	Ring indicate
DRS_ON	DRS_OFF	Data signal rate select

Table 12-5 V.28/RS-232C Interface Lead Transitions

V.35 Interface		
OFF to ON	ON to OFF	Description
RTS_ON	RTS_OFF	Request to send
CTS_ON	CTS_OFF	Clear to send
DSR_ON	DSR_OFF	Data set ready
CD_ON	CD_OFF	Carrier detect
DTR_ON	DTR_OFF	Data terminal ready
RI_ON	RI_OFF	Ring indicate

Table 12-6 V.35 Interface Lead Transitions

V.36/RS-449 Interface		
OFF to ON	ON to OFF	Description
RTS_ON	RTS_OFF	Request to send
CTS_ON	CTS_OFF	Clear to send
DSR_ON	DSR_OFF	Data set ready
DTR_ON	DTR_OFF	Data terminal ready
RI_ON	RI_OFF	Calling indicator
DRS_ON	DRS_OFF	Data signal rate select
CD_ON	CD_OFF	Data channel received line signal

Table 12-7 V.36/RS-449 Interface Lead Transitions

Protocol Actions

Messages can be transmitted using any of the commands described in Section 11. The test manager and/or the automatic protocol state machine provides the response.

Using Buffers

IDACOM's test manager has 256 buffers available for creating customized frames. These buffers are numbered from 0 through 255 and can be created any size desired. However, the BSC Emulation limits the number of bytes that can be transmitted in a message block to 5000.

A buffer consists of four bytes with values of 0, two bytes containing the length of the text, and the remaining bytes consisting of user-defined text.

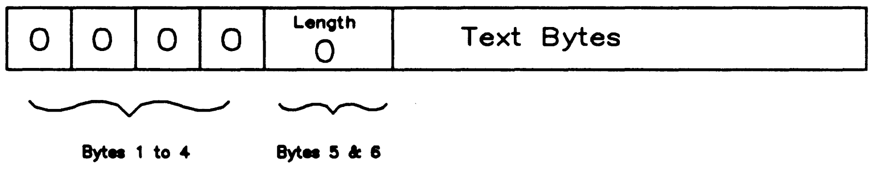


Figure 12-1 Buffer Structure

NOTE
All buffers are cleared when the TCLR command is issued. TCLR is usually the first command compiled when loading a test script.

There are three methods of moving text into a buffer.

Methods 1 and 2 automatically allocate memory for the specified text. Method 3 requires the user to allocate memory before moving text into the buffer. Use the TCLR command to clear all buffers.

Method 1

STRING->BUFFER (string\buffer number --)
Loads a quoted string into the specified buffer. The length is limited to 80 bytes if typing directly on the keyboard and 255 bytes if used within a test script. Either an ASCII or hex string can be specified. Valid buffer numbers are 0 through 255.

Example:
```
" IDACOM"  1  STRING->BUFFER          ( ASCII text moved to Buffer #1 )
X" 0100100100434445" 2 STRING->BUFFER ( Hex string of 8 bytes moved to Buffer #2 )
```

Method 2

FILE->BUFFER (filename\buffer number --)
Transfers a text file into the specified buffer (for text greater than 80 bytes). The file is created using the Edit function available on the Home processor. At this time, only ASCII text can be created. The last character to be transferred should be followed immediately by a CTRL 'p' character in the file. This special character is displayed as a pilcrow (¶) character. The file is transferred into the buffer until the ASCII control 'p' character is found or until the end of the file.

Example:
Create a file with the name CUSTOM.F and transfer to Buffer #3.
```
" CUSTOM.F"  3  FILE->BUFFER
```

Method 3

The following commands should not be used with FILE->BUFFER or STRING->BUFFER.

ALLOT_BUFFER (size \ buffer number -- flag)
Allocates memory for the specified buffer. ALLOT_BUFFER returns 0 if an error occurred, or 1 if correct.

NOTE
ALLOT_BUFFER should not be used repetitively with the same buffer number in the same test script.

FILL_BUFFER (data address \ size \ buffer number --)
Moves data, of a specified size, into a buffer. Previous contents are overwritten.

APPEND_TO_BUFFER (data address \ size \ buffer number --)
Appends data, of a specified size, into a buffer.

CLEAR_BUFFER (buffer number --)
Stores a size of 0 in the buffer. CLEAR_BUFFER has no effect on the allocated memory defined with ALLOT_BUFFER.

Example:

```
0 VARIABLE tempstring 6 ALLOT
" A TEST " tempstring $!                ( Initialize the string )
16 3 ALLOT_BUFFER                       ( Allocate 16 bytes of memory )
IF
      tempstring 4+ 5 3 FILL_BUFFER     ( Move 'TEST ' to buffer )
      " FAIL" COUNT 3 APPEND_TO_BUFFER  ( Append 'FAIL' to buffer )
ENDIF
```

BUFFER (buffer number -- address | 0)
Returns the address of the first byte of the specified buffer. The buffer must have been previously created by FILE->BUFFER, STRING->BUFFER, or ALLOT_BUFFER. A '0' is returned when the buffer is not created or an invalid buffer number is specified. Valid buffer numbers are 0 through 255.

Sending a Buffer

The text must first be stored in the buffer with the STRING->BUFFER or FILE->BUFFER commands. Once the text is in place, the buffer can be transmitted repetitively.

The actual size of the message block sent is defined by the block size set on the Emulation Configuration Menu or by the value stored in MAXIMUM_DATA_BLOCK_SIZE.

SEND_BUFFER (buffer number--)
Transmits the specified buffer as an entire message. Valid buffer numbers are 0 through 255.

Example:
Create the text to be included in the buffer and transmit the buffer after receiving an ACK0 or ACK1 response.

```
5 STATE{
        ACK0 ACK1 2 ?RX
        ACTION{
            " This is a message."  2 STRING->BUFFER      ( Create buffer )
            2 SEND_BUFFER                                ( Send buffer )
        }ACTION
  }STATE
```

NOTE
The message is not enclosed with STX or ETX/ETB control characters.

Example:
Create the following text in EBCDIC hex format to be included in the buffer: STX (0x02), ESC (0x27), ERASE/WRITE command code (0xF5), text "HELLO" (0xC8C5D3D3D6), and ETX (0x03). Transmit the buffer after receiving an ACK0 response (SELECT sequence).

```
STATE{
     ACK0 1 ?RX
     ACTION{
           X" 0227F5C8C5D3D3D603" 2 STRING->BUFFER     ( Create buffer )
           2 SEND_BUFFER                               ( Send buffer contents )
      }ACTION
}STATE
```

NOTE
The BSC protocol character set determines whether the contents of the buffer are formatted in EBCDIC or ASCII.

SEND_BUFFER_ERROR (buffer number--)
This command is similar to SEND_BUFFER, except the CRC value is calculated incorrectly.

13 TEST SCRIPTS

This section contains sample complete test scripts. These test scripts have also been supplied on disk and can be loaded and run as described in the Programmer's Reference Manual.

13.1 3780.F

The 3780.F test script simulates data transmission on a 3780 link. Several important features of 3780 are not implemented, including timeouts and reverse interrupts.

```
( This 3780 emulation script will automatically respond to line bids.   )
( To transmit data message use the following keys:                      )
(              UF1 - Begin repeated transmission of data                 )
(              UF2 - Stop transmission of data                           )

REP_COMP                                ( Select complete format report )
BSC-ASCII                               ( Select text report to be ASCII )
3270_DIS                                ( Disable 3270 format report reporting )

( Set up the data message )

" 0X7777 IDACOM ELECTRONICS LTD BRINGS TO YOU THE PT300 !!!! "      SET_MESSAGE
" THE PROTOCOL TESTER THAT LEADS THE WAY INTO THE FUTURE OF "       ADD_TEXT
" DATA COMMUNICATIONS "                                             ADD_TEXT
" NOW A TRUE 'FORTH' GENERATION COMPUTER IS AVAILABLE TO ASSIST "   ADD_TEXT
" WITH SOLVING PROTOCOL PROBLEMS."                                  ADD_TEXT

0 STATE_INIT{
       " Begin " 1 LABEL_KEY                  ( Define function key UF1. )
       " Stop "  2 LABEL_KEY                  ( Define function key UF2. )
  }STATE_INIT
```

```
0 STATE{                                    ( Idle state )
     ENQ 1 ?RX                              ( Is the event a line-bid ? i.e. ENQ )
         ACTION{                            ( Yes: response with a positive )
             ACK0_TX                        (       acknowledgement )
             T." Station became Slave" TCR  ( Display emulation is slave )
             1 NEW_STATE                    ( then enter state 1 )
         }ACTION
     UF1 ?KEY                               ( Is the event a UF1 function-key? )
         ACTION{                            ( Yes: response with an ENQ char )
             ENQ_TX
             T." Station became Master" TCR ( Display emulation is master )
             3 NEW_STATE                    ( Enter into state 3 )
         }ACTION
  }STATE

1 STATE{                                 ( RX ODD state )
     ETB_DATA ETX_DATA 2 ?RX             ( Detecting message block event )
         ACTION{
             ACK1_TX                     ( If message block event, response with )
             2 NEW_STATE                 ( ACK1 then enter into state 2 )
         }ACTION
     ENQ 1 ?RX                           ( Detecting ENQ event )
         ACTION{
             ACK0_TX                     ( If ENQ event, response with an ACK0 )
         }ACTION
     EOT 1 ?RX                           ( Detecting EOT event )
         ACTION{
             T." Line relinquished by master." TCR   ( Display link terminated )
             0 NEW_STATE                             ( by master unit then     )
         }ACTION                                     ( enter into state 0.     )
  }STATE

2 STATE{                                 ( RX EVEN state )
     ETB_DATA ETX_DATA 2 ?RX             ( Detecting message block event )
         ACTION{
             ACK0_TX                     ( If message block event, response with )
             1 NEW_STATE                 ( ACK0 then enter into state 1 )
         }ACTION
     ENQ 1 ?RX                           ( Detecting ENQ event )
         ACTION{
             ACK1_TX                     ( If ENQ event, response with an ACK0 )
         }ACTION
     EOT 1 ?RX                           ( Detecting EOT event )
         ACTION{
             T." Line relinquished by master." TCR   ( Display link terminated  )
             0 NEW_STATE                             ( by slave unit then enter )
         }ACTION                                     ( into state 0.    )
  }STATE
```

```
3 STATE{                                    ( TX EVEN state )
    ACK0 1 ?RX                              ( Detecting ACK0 event )
        ACTION{                             ( If ACK0 event: get address & size  )
            FIRST_BLOCK BSC_TX              ( of first message block of message to )
            4 NEW_STATE                     ( send then enter into state 4.       )
        }ACTION
    UF2 ?KEY                                ( Detecting a UF2 function-key event )
        ACTION{                                     ( If UF2 event, then display )
            T." Transmission being stopped." TCR    ( termination of data trans- )
            5 NEW_STATE                             ( mission notice and enter   )
        }ACTION                                     ( into state 5.  )
  }STATE

4 STATE{                                    ( TX ODD state )
    ACK1 1 ?RX                              ( Detecting ACK1 event )
        ACTION{                             ( If ACK1 event: get address & size )
            FIRST_BLOCK BSC_TX              ( of first message block of message to
            3 NEW_STATE                     ( send then enter into state 3.      )
        }ACTION
    UF2 ?KEY                                ( Detecting a UF2 function-key event )
        ACTION{                                 ( If UF2 event: display )
            T." Transmission being stopped." TCR  ( termination of data trans- )
            6 NEW_STATE                         ( mission notice and enter   )
        }ACTION                                 ( into state 6.  )
  }STATE

5 STATE{
    ACK0 1 ?RX                              ( Detecting ACK0 event )
        ACTION{
            T." Line relinquished." TCR     ( Display link termination and     )
            EOT_TX                          ( send an EOT control character,   )
            0 NEW_STATE                     ( then enter into state 0 )
        }ACTION                             ( Neither end is designated as the )
  }STATE                                    ( master or the slave unit )

6 STATE{
    ACK1 1 ?RX                              ( Detecting ACK1 event )
        ACTION{                             ( Display link termination and  )
            T." Line relinquished." TCR     ( send an EOT control character, )
            EOT_TX                          ( then enter into state 0 )
            0 NEW_STATE                     ( Neither end is designated as the )
        }ACTION                             ( master or the slave unit )
  }STATE
```

13.2 SAMPLE1.F

The SAMPLE1.F test script counts the number of general polls on a link using the variable counter. When a function key is pressed:

- the number of polls are displayed (since the key was pressed last);
- the time is displayed; and
- the counter is set to zero.

The output is displayed in the Data Window.

```
( File Title: SAMPLE1.F                                         )
(                                                               )
( This script counts the number of General Polls that a cluster )
( has received since the last time that UF1 has been pressed.   )

TCLR

0 STATE_INIT{
      " Exec" 1 LABEL_KEY                 ( Label function-key UF1 )
  }STATE_INIT

0 STATE{
      GENERAL_POLL 1 ?RX                   ( When a General poll is received, )
          ACTION{                          ( increment counter by 1 )
              1 COUNTER +!
          }ACTION
      UF1 ?KEY                             ( When the UF1 function is pressed, )
          ACTION{
              T." Number of General Polls Received = "  ( display the number of  )
              COUNTER @ T.  TCR                          ( received General poll, )
              T." Time = "
              GET_TIME T. T. T. T. T. T. TCR             ( display time, and      )
              0 COUNTER !                                ( reset counter to zero. )
          }ACTION
  }STATE
```

13.3 SAMPLE2.F

The SAMPLE2.F test script counts the number of general polls for three different cluster controllers which have logical unit devices 0, 7, and 13 (poll cluster address hex 40, hex C7, and hex 4D) activated. The output is displayed in the Data Window.

```
( FILE TITLE: SAMPLE2.F                                           )
(                                                                 )
( This script counts the General Polls received by the each of the 3 )
( specified clusters and prints out the total when UF1 is executed.  )
TCLR                                    ( Clear Test Manager memory )

0 VARIABLE CLUST_0                      ( Define counter for controller 1 )
0 VARIABLE CLUST_7                      ( Define counter for controller 2 )
0 VARIABLE CLUST_13                     ( Define counter for controller 3 )
0 STATE_INIT{
        " Exec" 1 LABEL_KEY             ( Label UF1 function-key )
  }STATE_INIT

0 STATE{
      GENERAL_POLL 1 ?RX                ( When a General poll is received )
          ACTION{
              LogicalUnit @             ( determine which controller unit )
              DOCASE                    ( is polled and increment        )
                  CASE 0   { 1 CLUST_0 +! }   ( the counter of that     )
                  CASE 7   { 1 CLUST_7 +! }   ( controller unit.        )
                  CASE 13  { 1 CLUST_13 +! }  ( If none of the three    )
                  CASE DUP { }          ( controller or polled,          )
              ENDCASE                   ( do nothing.                    )
          }ACTION
          UF1 ?KEY                      ( When function-key UF1 is )
          ACTION{                       ( received, check if emulation )
              EMULATION_TYPE @ CLUSTER =   ( is a CLST.               )
              IF                        ( If emulation = CLST )
                  T." Number of General Polls received "
                  TCR                               ( display numbers of   )
                  T." Cluster  0 = " CLUST_0  @ T. TCR    ( polls for cluster 0, )
                  T." Cluster  7 = " CLUST_7  @ T. TCR    ( cluster 7,  cluster  )
                  T." Cluster 13 = " CLUST_13 @ T. TCR    ( 13, and the time. )
                  T." Time = " GET_TIME T. T. T. T. T. T.
                  TCR
                  0 CLUST_0 !                       ( Reset all cluster )
                  0 CLUST_7 !                       ( counter to zero   )
                  0 CLUST_13 !
              ELSE
                  " Script valid only when run on CLUSTER side "
                  W.NOTICE                          ( If emulation = COMM )
              THEN                                  ( display the notice  )
          }ACTION                                   ( and do nothing      )
  }STATE
```

13.4 SAMPLE3A.F

The SAMPLE3A.F test script is the same as SAMPLE2.F except that the output is displayed in the User Window.

```
( File Title:  SAMPLE3A.F                                               )
(                                                                       )
(         Output from test script are written to section                )
(         of the screen called USER Window.                             )
(                                                                       )
( When the function key UF1 is pressed, the statistics are displayed on )
( the screen and are scrolled off when the screen is full. To see the   )
( monitor data display again, use the SHOW_DATA command. To see the User )
( Window again use the SHOW_USER command.                               )

TCLR                                   ( Clear Test Manager's memory )

#IFNOTDEF CLUST_0                      ( Define counters for controller )
    0 VARIABLE CLUST_0                 ( 0, 7, 13, if not defined and   )
    0 VARIABLE CLUST_7                 ( set them to zero.  )
    0 VARIABLE CLUST_13
#ENDIF
0 STATE{                               ( If General poll is received,   )
      GENERAL_POLL 1 ?RX               ( determine which controller is  )
      ACTION{                          ( polled. If controller 0, 7, or )
          LogicalUnit @                ( 13 is polled then increment    )
          DOCASE                       ( the polled controller counter. )
              CASE 0
                     { 1 CLUST_0 +! }
              CASE 7
                     { 1 CLUST_7 +! }
              CASE 13
                     { 1 CLUST_13 +! }
              CASE DUP { }             ( If it is for neither )
          ENDCASE                      ( controller of interest, then   )
      }ACTION                          ( do nothing.    )
      UF1 ?KEY                         ( Determine if a function-key is pressed )
      ACTION{
          POP_USER                          ( If Yes, open User Window and    )
              W." Cluster 0 " CLUST_0 @ W.  ( display polled statistic of     )
              W." Cluster 7 " CLUST_7 @ W.  ( all controller counter to User )
              W." Cluster 13 " CLUST_13 @ W. ( Window. Also display the time  )
              W." Time: "                   ( when the function-key was       )
              GET_TIME W. W. W. W. W. W.    ( pressed.                        )
              WCR
              0 CLUST_0 !                   ( Reset counters to zero. )
              0 CLUST_7 !
              0 CLUST_13 !
          CLOSE_WINDOW                      ( Close User Window. )
      }ACTION
  }STATE
```

```
OPEN_USER                               ( Open  User Window clear text  )
      CLEAR_TEXT                        ( from User Window and create a )
      RED_BG PAINT                      ( red background, then close    )
CLOSE_WINDOW                            ( User Window. )
```

13.5 MON_SCRIPT.F

The MON_SCRIPT.F test script displays bar graphs in the User Window. Bar graphs represent statistics gathered from a BSC line.

```
(                                                                      )
(    File Title: MON_SCRIPT.F                                          )
(                                                                      )
(    The statistics are maintained in two sets of variables. One set, ending )
(    with _SHORT, is displayed every second and the other, ending with )
(    _LONG, is displayed every minute.                                 )
(                                                                      )
(         UF1: displays the bar graph                                  )
(         UF2: displays the data Window                                )

    TCLR

    0 VARIABLE POLL_COUNT_SHORT          ( Define Short counters for received    )
    0 VARIABLE DATA_COUNT_SHORT          ( Select polls, General/Specific polls, )
    0 VARIABLE NAK_COUNT_SHORT           ( NAK, and data stream message size.    )
    0 VARIABLE SELECT_COUNT_SHORT

    0 VARIABLE POLL_COUNT_LONG           ( Define Long counters for Select,  )
    0 VARIABLE NAK_COUNT_LONG            ( polls, General/Specific polls, and )
    0 VARIABLE SELECT_COUNT_LONG         ( NAK. )


    0 SEQ[  ( -- )                       ( Initialize Short counters    )
            0 POLL_COUNT_SHORT !         ( to zero and start timer 1.   )
            0 DATA_COUNT_SHORT !
            0 NAK_COUNT_SHORT !
            0 SELECT_COUNT_SHORT !
            1 10 START_TIMER
      ]SEQ

    1 SEQ[
            0 POLL_COUNT_LONG !          ( Initialize Long counters )
            0 NAK_COUNT_LONG !           ( zero and start timer 2.  )
            0 SELECT_COUNT_LONG !
            2 600 START_TIMER
    ]SEQ

    0 VARIABLE COLOR

    2 SEQ[  ( amount  " label" --- )        ( Sequence to display a bar graph. )
    WHERE DROP CLEAR_ROW WHERE
    DROP WHI_FG PAINT_ROW                ( Clear this row )
    COUNT   W.TYPE                       ( Display the label )
    WHERE DROP 12 THERE W." :"           ( Put a colon at column 12 )
    DUP W.                               ( Print the value )
    DUP 50 >                             ( Determine the color )
    IF
        RED_BG COLOR !                   ( Red if over 50 )
        DROP 50
    ELSE
        CYA_BG COLOR !                   ( Green if below 50 )
    ENDIF
    WHERE DROP 18 THERE                  ( Move cursor to position 18 )
    " . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ."
    1+ SWAP
    COLOR @ W.TYPE_A                     ( Display the bar graph )
 }SEQ

0 STATE_INIT{
    " Graph" 1 LABEL_KEY                 ( Label Function-key UF1 )
    " Data" 2 LABEL_KEY                  ( Label Function-key UF2 )
    POP_USER                             ( Open User Window and display graph )
    DISPLAY_STATUS_LINE                  ( heading, graph labels, bar graph )
    CLEAR_TEXT 0 PAINT                       ( values, reset counters to   )
    1 30 THERE  " 3270 Analysis Script"      ( zero, start timer, and dis- )
    COUNT BLU_BG W.TYPE_A                    ( able write to User Window.  )
    2 26 THERE " Short Time Interval is 1 Second"
    COUNT BLU_BG W.TYPE_A
    4 18 THERE
    " ----5----10---15---20---25---30---35---40---45---50"
    COUNT GRN_FG W.TYPE_A
    9 26 THERE "  Long Time Interval is 1 Minute"
    COUNT BLU_BG W.TYPE_A
    0 RUN_SEQ 1 RUN_SEQ
    CLOSE_WINDOW
}STATE_INIT

0 STATE{
        GENERAL_POLL SPECIFIC_POLL 2 ?RX     ( If a General or Specific    )
        ACTION{                              ( poll is received increment )
                1 POLL_COUNT_LONG +!         ( both Long poll counter and )
                1 POLL_COUNT_SHORT +!        ( Short poll counter. )
        }ACTION
        SELECT 1 ?RX
        ACTION{                              ( If a Select poll is        )
                1 SELECT_COUNT_SHORT +!      ( received increment the     )
                1 SELECT_COUNT_LONG +!       ( select Short counter and   )
        }ACTION                              ( Select Long counter.       )
        NAK 1 ?RX                            ( If a NAK is received, in- )
        ACTION{                         ( crement both NAK Short     )
                1 NAK_COUNT_LONG +!     ( and Long poll counter.     )
                1 NAK_COUNT_SHORT +!
        }ACTION
        ETB_DATA ETX_DATA 2 ?RX         ( If data transfer occurs,      )
        ACTION{                         ( determine length of received )
                L3-LENGTH @             ( data, add length to current  )
                DATA_COUNT_SHORT +!     ( accumulated length and       )
        }ACTION                         ( save it. )

        1 ?TIMER                        ( Displays graphical representation )
        ACTION{                         ( of the Short statistical data. )
               OPEN_USER                ( Enable write to User Window. )
               5 0 THERE POLL_COUNT_SHORT @   " Polls"        2 RUN_SEQ
               6 0 THERE NAK_COUNT_SHORT @    " Naks"         2 RUN_SEQ
               7 0 THERE DATA_COUNT_SHORT @   " Data bytes"   2 RUN_SEQ
               8 0 THERE SELECT_COUNT_SHORT @ " Selects"      2 RUN_SEQ
               0 RUN_SEQ
              CLOSE_WINDOW              ( Disable write to User Window. )
        }ACTION
        2 ?TIMER                        ( Displays graphical representation )
        ACTION{                         ( of the Long statistical data. )
              OPEN_USER                 ( Enable write to User Window )
              10 0 THERE POLL_COUNT_LONG @ 60 /
                   " Avg Polls/S" 2 RUN_SEQ
              11 0 THERE NAK_COUNT_LONG @
                   " Naks/Min"    2 RUN_SEQ
              12 0 THERE SELECT_COUNT_LONG @
                   " Selects/Min" 2 RUN_SEQ
              1 RUN_SEQ
              CLOSE_WINDOW              ( Disable write to User Window. )
        }ACTION
        UF1 ?KEY                        ( Display User Window when      )
        ACTION{                         ( function-key UF1 is pressed.  )
                SHOW_USER
                DISPLAY_STATUS_LINE
        }ACTION
        UF2 ?KEY                        ( Display Data Window when      )
        ACTION{                         ( function-key UF2 is pressed. )
                SHOW_DATA
                DISPLAY_STATUS_LINE
        }ACTION
}STATE
```

13.6 MON_BAR1.F

The MON_BAR1.F test script displays the following vertical bar graphs in the User Window:

- Poll distribution
- Traffic distribution
- Outbound traffic
- Inbound traffic

The poll distribution graph (default) displays the amount of polling (request/response) between the host system and each existing cluster controller within the specified time interval. The percentage is calculated by the following equation:

$$\text{Percentage Polled per CU} = \frac{\text{no. of poll per each CU}}{\text{total no. of poll per time interval}} \times 100\%$$

This test script automatically determines whether the host system is a DCE or DTE. The test script then enters into the statistic gathering states to accumulate the appropriate data used for calculating the values in the four graphs.

The traffic distribution graph displays the total traffic (inbound and outbound data) distribution of the total communication line usage between the host system and each existing cluster controller within the specified time interval. The percentage is calculated by the following equation:

$$\text{Percentage Traffic Distribution} = \frac{\text{total bytes transferred per TCU+CU}}{\text{total bytes transferred per interval}} \times 100\%$$

The outbound traffic graph displays the data traffic (outbound) distribution of the total communication line usage between the host system and each existing cluster controller within the specified time interval. The percentage is calculated by the following equation:

$$\text{Percentage of Outbount Traffic per CU} = \frac{\text{total received bytes per CU}}{\text{total bytes transferred per TCU+CU}} \times 100\%$$

The inbound traffic graph displays the data (inbound) traffic distribution of the total communication line usage between the host system and each existing cluster controller within the specified time interval. The percentage is calculated by the following equation:

$$\text{Percentage of Inbound Traffic per CU} = \frac{\text{total sent bytes per CU}}{\text{total bytes transferred per TCU+CU}} \times 100\%$$

The default time interval is three seconds and can be changed to any other value using the TIME–INT variable. When changing this value, the new time interval value must be entered in tenths of seconds.

Example:
Set the time interval to 43.6 seconds.

```
436 TIME-INT !
```

For an accurate representation of the actual line utilization, use a TIME–INT value greater than one minute (i.e. 600). With a large time interval, the displayed graph values are averaged out throughout the time interval.

```
( File Title:  MON_BAR1.F                               )
(                                                       )
( To be used on top of BSC MONITOR program              )
( LINE UTILIZATION BAR GRAPH:  - Poll Distribution      )
(                              - Traffic Distribution   )
(                              - Outbound Traffic per CU )
(                              - Inbound Traffic per CU  )

    TCLR
  #IFNOTDEF TIME-INT

  50 VARIABLE TIME-INT                 ( Time interval variable              )
  0 VARIABLE COLOUR                    ( Contains bar graph colors info     )
  0 VARIABLE %PERCENT                  ( Contains % value per CU             )
  0 VARIABLE COLMN                     ( Contains graph column position info )
  0 VARIABLE HOLD-LU                   ( Contains valid CU logical unit no . )
  0 VARIABLE DATA-FLOW                 ( Determines if COMM = DTE or DCE     )
  0 VARIABLE SCALAR                    ( Indicates if graph scaling          )
  0 VARIABLE TSPOL                     ( Total no. of poll sequences         )
  0 VARIABLE LU-SAVE                   ( Variable contains last valid LU     )
  0 VARIABLE CLUST-POLL 32 ALLOTL      ( Contain no. of polls per cluster    )
  0 VARIABLE PLOT-TYPE                 ( Indicate plot type  displayed       )
  0 VARIABLE RU-TOT-LENGTH             ( Total bytes transferred per time interval  )
  0 VARIABLE RU-OUT 32 ALLOTL          ( Outbound bytes sent per CU         )
  0 VARIABLE RU-IN 32 ALLOTL           ( Inbound bytes received per CU       )
  0 VARIABLE RU-TOT 32 ALLOTL          ( Total bytes transferred per CU      )

 : VERTICAL  ( -- )                    ( Draws and labels graph's Y- axis    )
      POP_USER  SCALAR @ 0=            ( Check graph scaling flag            )
      IF  15 3                         ( Down-scaling is not wanted          )
          DO I 10 THERE "   " COUNT    ( Erase old Y-axis labels             )
          BLK_BG W.TYPE_A LOOP
          15 3 DO
          I DUP DUP 9 >                ( Label Y-axis )
          IF 11 ELSE 10 ENDIF
          THERE 14 SWAP -
          << W.  " |" COUNT
          BLU_FG W.TYPE_A
          LOOP
      ELSE  15 3                              ( Graph down-scaling wanted          )
          DO I DUP DUP 12 >                   ( Label Y-axis    )
          IF 11 ELSE 10 ENDIF
          THERE 14 SWAP - 3 <<# W. " |"
          COUNT BLU_FG W.TYPE_A
          LOOP
      ENDIF
   CLOSE_WINDOW
 ;

 ( VPAINT performs % calculation for each cluster and plots the )
 ( corresponding bar graph for each cluster being polled. )

 : VPAINT ( -- )                              ( Check if down-scaling is wanted )
     %PERCENT @ SCALAR @
     IF  2 >># 22 >                           ( Yes:  down-scaling is wanted     )
        IF 13 ELSE %PERCENT @ 3 >># ENDIF
     ELSE 22 >
        IF 13 ELSE %PERCENT @ >> ENDIF
     ENDIF
     15 - ABS DUP 1 - COLMN @ THERE           ( Display % value on bar top       )
     %PERCENT @ DUP 100 >
      IF DROP 100 ENDIF W.                    ( Clamp to maximum of 100%         )
      15 SWAP
      DO                                      ( Paint column with selected color )
        I DUP 14 >
        IF
           %PERCENT @ 1 <
           IF
              DROP 14 COLMN @ THERE
              " __"  COUNT BLU_FG W.TYPE_A
           ELSE DROP
           ENDIF
        ELSE
           COLMN @ THERE
           "    " COUNT COLOUR @ W.TYPE_A
        ENDIF
      LOOP
;

( NO_PAINT  erases the old paint and repair X-axis that )
(       has been damaged during the erase routine.      )
```

```
: NO_PAINT  ( -- )                         ( Erase selected painted columns )
     15 1
     DO I DUP COLMN @ THERE                ( Erase  painted column    )
        "    " COUNT BLK_BG W.TYPE_A
        14  =                              ( Repair X-axis if erased )
        IF 14 COLMN @
           THERE " ___" COUNT
           BLU_FG W.TYPE_A
        ENDIF
    LOOP
  VPAINT
;
: POLL_DIST ( clust -- )
     2 <<# CLUST-POLL + @                 ( Adjust array pointer     )
     100 * TSPOL @                        ( Calculate percentage     )
     DIVIDE %PERCENT !
     NO_PAINT                             ( Draw bar graph           )
;
: TRAFFIC_DIST ( clust -- )
     2 <<# DUP RU-OUT + @                 ( Adjust array pointers    )
     SWAP RU-IN + @ +
     100 * RU-TOT-LENGTH @                ( Calculate traffic dist. percentage  )
     DIVIDE %PERCENT !
     NO_PAINT                             ( Draw bar graph         )
;
: OUTBOUND ( clust -- )
     2 <<# DUP RU-OUT + @                 ( Adjust array pointer )
     100 * SWAP RU-TOT + @                ( Calculate outbound data percentage )
     DIVIDE %PERCENT !
     NO_PAINT                             ( Draw bar graph        )
;
: INBOUND ( clust -- )
     2 <<# DUP RU-IN + @                  ( Adjust array pointer )
     100 * SWAP RU-TOT + @                ( Calculate inbound data percentage )
     DIVIDE %PERCENT !
     NO_PAINT                             ( Draw bar graph                    )
;
: GRAPH_TYPE ( clust/attr -- )
     COLOUR ! DUP 3 * 14 + COLMN !        ( Stores clst bar position & attr    )
     PLOT-TYPE @
     DOCASE                               ( and determines graph type wanted. )
         CASE  0    { POLL_DIST }
         CASE  1    { TRAFFIC_DIST }
         CASE  2    { OUTBOUND   }
         CASE  3    { INBOUND    }
         CASE DUP   { POLL_DIST }
     ENDCASE
;
```

```
: VALID_LU  ( -- clust )                   ( Check if incoming messages has   )
    HOLD-LU @
    LogicalUnit @ DUP -1 =                 ( valid header containing cluster  )
    ROT OR                                 ( controller address.              )
    IF
       DROP LU-SAVE @
    ELSE
       DUP LU-SAVE !
    ENDIF
;
: HEADING ( -- )                           ( Enable write to User Window and )
    POP_USER                               ( writes the appropriate heading   )
    DISPLAY_STATUS_LINE                    ( for the graph to be display.    )
    0 34 THERE
    PLOT-TYPE @
    DOCASE
         CASE  0  { " POLLING DISTRIBUTION " }
         CASE  1  { " TRAFFIC DISTRIBUTION " }
         CASE  2  { " OUTBOUND TRAFFIC / CU" }
         CASE  3  { " INBOUND TRAFFIC / CU " }
         CASE DUP { " POLLING DISTRIBUTION " }
    ENDCASE
    COUNT BLU_BG W.TYPE_A
    CLOSE_WINDOW
;
: Y_AXIS ( -- )                            ( Enable write to User Window )
    POP_USER                               ( then label the Y-axis of the )
    6 0 THERE                              ( graph appropriately, then )
    PLOT-TYPE @                            ( disable write to User Window.)
    DOCASE
       CASE  0  { "  POLL  " }
       CASE  1  { " TRAFFIC" }
       CASE  2  { " OUT_BND" }
       CASE  3  { " IN_BND " }
       CASE DUP { "  POLL  " }
    ENDCASE
    COUNT GRN_FG W.TYPE_A
  CLOSE_WINDOW
;
#ENDIF

  0 SEQ{  ( -- )                           ( Initialize counters & start timer  )
           0 TSPOL !                       ( -reset total poll within interval  )
           0 RU-TOT-LENGTH !               ( -empty received byte length        )
           CLUST-POLL 128 0 FILL           ( -empty # of polls per CU array     )
           RU-OUT 128 0 FILL               ( -empty outbound byte length array  )
           RU-IN 128 0 FILL
           RU-TOT 128 0 FILL
           1 TIME-INT @ START_TIMER
           2 2 START_TIMER
    }SEQ
```

```
1 SEQ[   ( -- )                             ( Define cluster units  attributes )
            OPEN_USER
         0  BLU_BG GRAPH_TYPE
         1  MAG_BG GRAPH_TYPE
         2  RED_BG GRAPH_TYPE
         3  CYA_BG GRAPH_TYPE
         4  BLU_BG GRAPH_TYPE
         5  MAG_BG GRAPH_TYPE
         6  RED_BG GRAPH_TYPE
         7  CYA_BG GRAPH_TYPE
         8  BLU_BG GRAPH_TYPE
         9  MAG_BG GRAPH_TYPE
        10  RED_BG GRAPH_TYPE
        11  CYA_BG GRAPH_TYPE
        12  BLU_BG GRAPH_TYPE
        13  MAG_BG GRAPH_TYPE
        14  RED_BG GRAPH_TYPE
        15  CYA_BG GRAPH_TYPE
          CLOSE_WINDOW

  ]SEQ
2 SEQ[                                      ( Starts TIME-INT update sequence. )
     2 10 START_TIMER
  ]SEQ
3 SEQ[ ( scalar/type -- )                   ( Determine if Y-axis down-scaling )
     PLOT-TYPE !                            ( is wanted, and the appropriate )
     SCALAR !                               ( Y-axis label. )
     HEADING Y_AXIS
     VERTICAL 1 RUN_SEQ
     SHOW_USER
 ]SEQ
 4 SEQ[
          REC-LENGTH @ DUP                  ( Get received frame length and    )
          RU-TOT-LENGTH +! DUP              ( adds it to IN/OUT data variable. )
          VALID_LU 2 <<# RU-TOT + +!
          VALID_LU 2 <<#
          DATA-FLOW @                       ( Does COMM = DCE side?   )
          IF                                ( Yes, COMM = DCE side.   )
              PORT-ID @  0xFF AND           ( Is received frame from DTE side. )
              TO_DCE_RX =
              IF                            ( COMM = DCE side & frame )
                  RU-OUT + +!               ( received from DTE side  )
              ELSE                          ( COMM = DTE side & frame )
                  RU-IN + +!                ( received from DCE side  )
              ENDIF
         ELSE                               ( No, COMM = DTE side. )
              PORT-ID @  0xFF AND           ( Is received frame from DCE side? )
              TO_DTE_RX =
              IF                            ( COMM = DTE side & frame )
                  RU-OUT + +!               ( received from DCE side  )
                  ELSE                          ( COMM = DCE side & frame )
                      RU-IN + +!                ( received from DTE side   )
                  ENDIF
             ENDIF
    }SEQ
    5 SEQ{ ( -- ) OPEN_USER                     ( Calculate line utilization percentage )
         RU-TOT-LENGTH @ 80 M*                  ( from the total data through-put and   )
         TIME-INT @ DIVIDE 50 *                 ( the time interval. )
         INTERFACE-SPEED @ DIVIDE
         DUP 99 >
         IF DROP 99 ENDIF                       ( Clamp to max. of 99% )
         0 74 THERE "      " COUNT BLK_BG W.TYPE_A
         0 74 THERE W. " %" COUNT GRN_FG W.TYPE_A
         CLOSE_WINDOW
    }SEQ
0 STATE_INIT{
        " DATA"      1 LABEL_KEY                ( Label user function-key )
        " POLL"      2 LABEL_KEY
        " TRAFFIC"   3 LABEL_KEY
        " OUT_BND"   4 LABEL_KEY
        " IN_BND"    5 LABEL_KEY
        " ACTIVE"    6 LABEL_KEY
        POP_USER                                ( Enable write to User Window )
        DISPLAY_STATUS_LINE                     ( Clear screen )
        CLEAR_TEXT 0 PAINT                      ( Display the default bar )
        0 34 THERE  " POLL DISTRIBUTION "       ( graph screen, POLL DIS- )
        COUNT BLU_BG W.TYPE_A                   ( TRIBUTION, and label the )
        14 14 THERE                             ( Y and X-axis appropriately )
                  "  ______________________________________________________  "
        COUNT BLU_FG W.TYPE_A
        15 3 THERE
         "      CU:     40 C1 C2 C3 C4 C5 C6 C7 C8 C9 4A 4B 4C 4D 4E 4F 50 51  "
        COUNT WHI_FG W.TYPE_A
        0 PLOT-TYPE !
        5 0 THERE "     %    " COUNT GRN_FG W.TYPE_A
        6 0 THERE "  POLL   " COUNT GRN_FG W.TYPE_A
        0 0 THERE " Time Int:" COUNT GRN_FG W.TYPE_A
        0 58 THERE "  % UTILIZATION:" COUNT GRN_FG W.TYPE_A
        2 RUN_SEQ 0 RUN_SEQ  CLOSE_WINDOW  VERTICAL
}STATE_INIT

0 STATE{
        GENERAL_POLL SPECIFIC_POLL
        SELECT 3 ?RX
        ACTION{                                 ( Determines if COMM = DTE or     )
              POP_USER                          ( COMM = DCE side and sets the    )
              1 0 THERE                         ( DATA-FLOW variable accordingly )
              PORT-ID @ 0xFF AND
              TO_DCE_RX =
              IF
                  1 DATA-FLOW !
                  " COMM: DCE"
              ELSE
                  0 DATA-FLOW !
                  " COMM: DTE"
              ENDIF
              COUNT GRN_FG W.TYPE_A
              CLOSE_WINDOW
              1 NEW_STATE
        }ACTION
  }STATE

 1 STATE{                                ( Displays real-time graphs of current values. )
    1 ?TIMER
          ACTION{
                5 RUN_SEQ
                1 RUN_SEQ                          ( Plot the wanted values.         )
                0 RUN_SEQ                          ( Reset all counters & arrays.    )
         }ACTION

    2 ?TIMER
          ACTION{  OPEN_USER
              0 10 THERE TIME-INT @ 10 M/ W.    ( Display time interval value.  )
              " ." COUNT CYA_FG W.TYPE_A W.
              " sec." COUNT GRN_FG W.TYPE_A
              2 RUN_SEQ CLOSE_WINDOW
          }ACTION

         GENERAL_POLL SPECIFIC_POLL 2 ?RX
         ACTION{                              ( Detects polling frames.       )
              0 HOLD-LU !
              1 TSPOL +!                      ( Increment poll counter.       )
              VALID_LU 2 <<# DUP DUP
              CLUST-POLL + 1 SWAP +!
              REC-LENGTH @ DUP                ( Get received frame length, add      )
              RU-TOT-LENGTH +! DUP            ( received length to previous IN/OUT )
              ROT RU-TOT + +!                 ( and OUT data variable/array.        )
              SWAP RU-OUT +  +!
         }ACTION

         SELECT 1 ?RX                         ( store cluster logical unit no.      )
         ACTION{
               0 HOLD-LU !
               4 RUN_SEQ
         }ACTION

         EOT ACK0 ACK1 NAK WACK RVI ENQ
         ETX_DATA ETB_DATA TTD                ( any message is received ?   )
         10 ?RX
         ACTION{
                L4-LENGTH @ 31 <
                IF 1 HOLD-LU ! ENDIF
                4 RUN_SEQ
         }ACTION
```

```
     UF1 ?KEY
          ACTION{
               SHOW_DATA                      ( Display Date Window )
          }ACTION
     UF2 ?KEY
          ACTION{
                0 0 3 RUN_SEQ                 ( Display POLL DISTRIBUTION graph )
          }ACTION
     UF3 ?KEY
          ACTION{
                0 1 3 RUN_SEQ                 ( Display TRAFFIC DISTRIBUTION graph )
          }ACTION
     UF4 ?KEY
          ACTION{
                1 2 3 RUN_SEQ                 ( Display OUTBOUND traffic graph )
          }ACTION
     UF5 ?KEY
          ACTION{
                1 3 3 RUN_SEQ                 ( Display INBOUND traffic graph )
          }ACTION
     UF6 ?KEY
          ACTION{                             ( Display non-realtime graphs )
               " INACTIVE" 6 LABEL_KEY
               2 NEW_STATE
          }ACTION
 }STATE

2 STATE{
     UF1 ?KEY
          ACTION{                             ( Display Data Window )
               SHOW_DATA
          }ACTION
     UF2 ?KEY
          ACTION{
                0 0 3 RUN_SEQ                 ( Display POLL DISTRIBUTION graph )
          }ACTION
     UF3 ?KEY
          ACTION{
                0 1 3 RUN_SEQ                 ( Display TRAFFIC DISTRIBUTION graph )
          }ACTION
     UF4 ?KEY
          ACTION{
                1 2 3 RUN_SEQ                 ( Display OUTBOUND traffic graph )
          }ACTION
     UF5 ?KEY
          ACTION{
                1 3 3 RUN_SEQ                 ( Display INBOUND traffic graph )
         }ACTION
     UF6 ?KEY
```

```
         ACTION[
               " ACTIVE" 6 LABEL_KEY        ( Display real-time graphs )
               0 RUN_SEQ
               2 RUN_SEQ
               1 NEW_STATE
         }ACTION
 }STATE
```

A
INTRODUCTION TO BISYNC

This appendix contains a brief introduction to BSC. For further information, the reader is advised to consult the CCITT Binary Information Synchronous Communication Recommendation, (Malaga–Torremolinos, 1984) and IBM 3270 Information Display System: 3274 Control Unit Description and Programmer's Guide, GA23–0061–2 publication.

A.1 BSC Configurations

There are two types of configurations for BSC:

- Point–to–point
- Multipoint

These configurations can operate in either two–way alternate (half duplex) or two–way simultaneous mode (full duplex). In two–way alternate mode, stations take turns transmitting. In two–way simultaneous mode, two stations can transmit and receive at the same time.

Point-to-Point

A point-to-point link connects two stations (the host system and the cluster controller). Links can be established on dedicated or dial-up circuits, and can be half duplex or full duplex.

Half·duplex channels can be switched (temporary connection) or non-switched (permanent connection).

There are three basic point-to-point data link configurations:

- Half duplex, non-switched
- Full duplex, non-switched
- Half duplex, switched

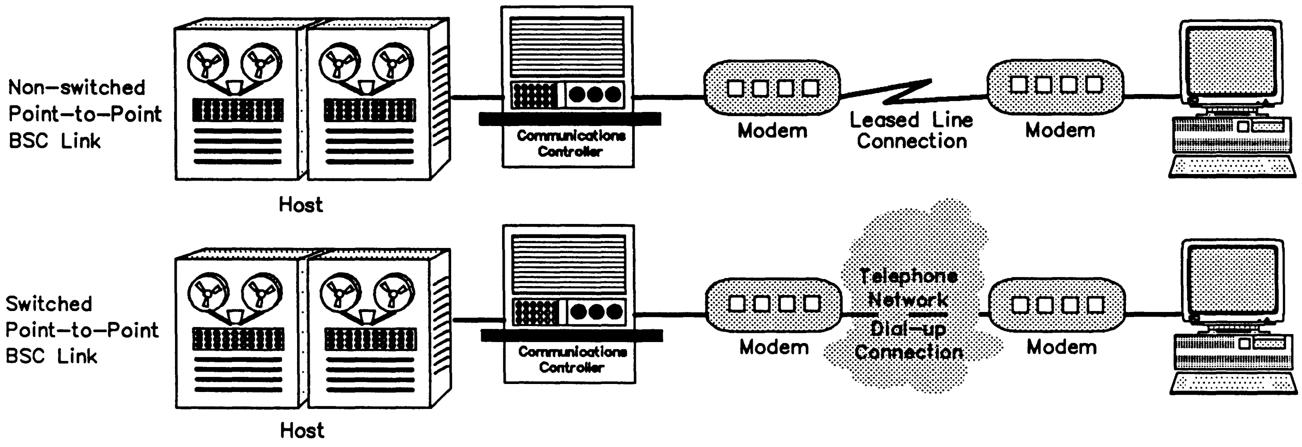


Figure A-1 Point-to-Point Configuration

Multipoint

A multipoint link connects more than two stations. Control procedures must be used to designate which stations can use the link at any one time. One station on the multidrop line is designated as the control station (host system), and all other stations (cluster controllers) are designated as tributaries. The control station acts as the traffic director, designating which tributaries use the link by the process of polling and selection. At any time, transmission on the multipoint link is between only two stations; all other stations on the link are in a passive receive mode. A full duplex channel can be used as two half duplex channels (eg. in some multipoint configurations, a station can transmit to one station while receiving from another station).

There are two basic multipoint data link configurations:

- Half duplex, non-switched
- Full duplex, non-switched

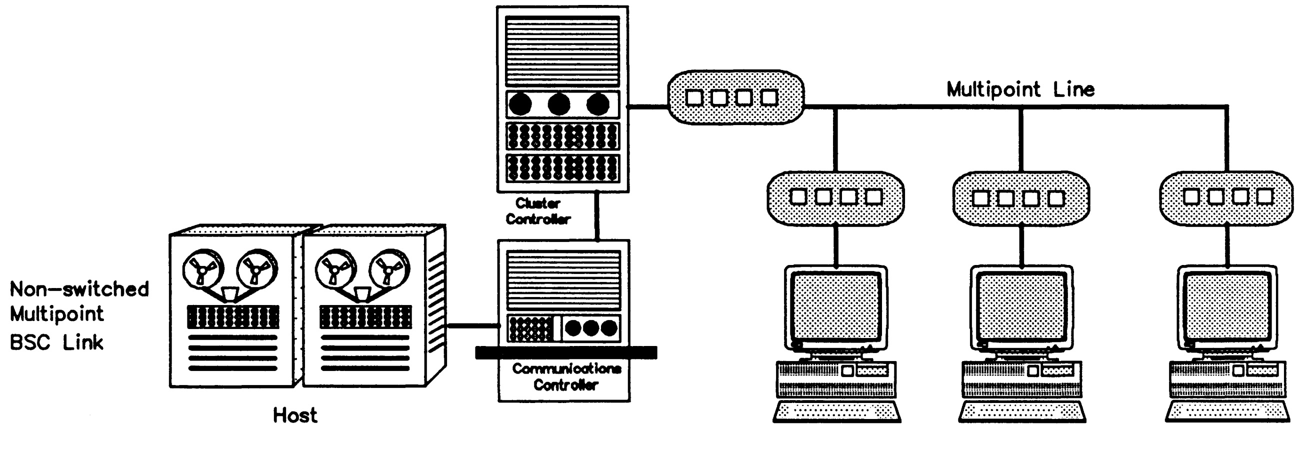


Figure A-2 Multipoint Configuration

A.2 Physical Layer

The physical layer provides physical, mechanical, and electrical conditions for the synchronous transmission of the bit streams generated by higher protocol layers. This is specified by reference to interface recommendations. CCITT recommends using the X.21 interface but many network implementations use V.28/RS-232C, V.36/RS-449 or V.35 for high speed transmissions. All of these physical interfaces are supported on IDACOM testers.

A.3 Message Block Structure

BSC messages consist of one or more blocks of text, called the body. The beginning of each block is identified by the STX control character, and all blocks (except the last block) are ended by an ETB or ITB control character. The last block of the message is ended by an ETX control character. These messages contain:

- an SYN (synchronous idle);
- an SOH (start of header);
- text; and
- a BCC (block check sequence).

NOTE
Some control characters (SYN, SOH, text, or BCC) may be missing when messages are used for control.

PAD	SYN	SYN	EOT	CU Addr	CU Addr	Gen Addr	Gen Addr	ENQ	PAD

Table A–1 General Poll Sequence Structure

PAD	SYN	SYN	EOT	CU Addr	CU Addr	Dev Addr	Dev Addr	ENQ	PAD

Table A–2 Specific Poll Sequence Structure

PAD	SYN	SYN	EOT	CU Addr	CU Addr	Dev Select Addr	Dev Select Addr	ENQ	PAD

Table A–3 Select Sequence Structure

PAD	SYN	SYN	STX	ESC	Write Com Code	WCC	Text	. . .	Text	ETX	BCC	PAD

Table A–4 Write Command Structure

PAD	SYN	SYN	STX	ESC	Copy Com Code	CCC	Text	. . .	Text	ETX	BCC	PAD

Table A–5 Copy Command Structure

PAD	SYN	SYN	STX	ESC	AID	Cursor Addr	Cursor Addr + 1	SBA	Attr Type	Attr Value	Order and/or Text	. . .
. . .	Order and/or Text	ETX	BCC	PAD								

Table A–6 Read Buffer Command Structure

PAD	SYN	SYN	SOH	%	R	STX	CU Poll Addr	Dev Addr	Status byte 1
Status byte 2	ETX	BCC	PAD						

Table A–7 Status Message Structure

PAD	SYN	SYN	SOH	%	/	STX	Text	. . .	Text	ETB /ETX	BCC	PAD

Table A–8 Test Request Message Structure

Code Structures

Each tributary station operates with one of three code structures:

- EBCDIC (Extended Binary Coded Decimal Interchange Code)
- ASCII (America Standards Code for Information Interchange)
- SBT (Six–Bit Transcode) – not supported

The choice of code depends on the application. For system compatibility, the code structure must be the same for all stations on the particular communication link.

Control Characters

Control characters provide reliable point–to–point and multipoint communications between the host system and remote terminals. Information is transferred across the communication link as data link control sequences, command messages, and application data messages. The access procedure used in BSC provides the following mechanisms:

- Initializes the link between the host system and device terminals
- Ensures any message can be transferred (transparency)
- Minimizes the probability of undetected errors
- Ensures the information is transferred across the link in the correct sequence
- Controls the flow of information across the link
- Detects and reports procedure errors
- Logically disconnects the link

Data link control is affected by properly recognizing control characters and taking appropriate actions. The following control characters are used in BSC:

SYN	**Synchronous Idle** Establishes and maintains synchronization between DCE's and acts as a filter between message blocks.
SOH	**Start of Header** Identifies the beginning of block header information.
STX	**Start of Text** Identifies the end of headers and the beginning of a block of text.
ETB	**End of Transmission Block** Identifies the end of a block started with SOH or STX. A BCC is sent immediately following an ETB.
ETX	**End of Text** Terminates a block of data started by an STX or SOH which was transmitted as an entity. A BCC is sent immediately following an ETX.
EOT	**End of Transmission** Indicates the end of a message transmission by this station. The message can contain one or more blocks. Receipt of an EOT causes all receiving stations to reset.
ENQ	**Enquiry** Requests a repeat transmission to a message block if the original response is garbled or not received.
ACK0 ACK1	**Affirmative Acknowledgment** Indicates the correct receipt of the last previous blocks, and that the receiver is ready to accept the next message block.

WACK	**Wait–Before–Transmit Affirmative Acknowledgment** Indicates a temporary not ready to receive condition to the sending station and affirmative acknowledgment of the last previous received message block.
NEG	**Negative Acknowledgment** Indicates that the last block was received in error and that the receiver is ready to receive another block.
DLE	**Data Link Escape** Indicates to the receiver that the character following DLE is to be interpreted as a control character.
RVI	**Reverse Interrupt** Indicates a request by a receiving station that the current transmission be terminated so a higher priority message can be sent.
TTD	**Temporary Text Delay** Indicates that the sending station is not ready to send immediately, but wants to keep the line.

Hexadecimal Representation			
BSC Control Character	ASCII Code	EBCDIC Code	SBT Code
SYN	16	32	3A
SOH	01	01	00
STX	02	02	0A
ETB	17	26	0F
ITB	1F	1F	1D
ETX	2E	03	2E
EOT	04	37	1E
ENQ	05	2D	2D
DLE	10	10	1F
ACK0	1000	1070	1F20
ACK1	1001	1061	1F20
NAK	15	3D	3D
WACK	1030	106B	1F26
RVI	103C	107C	1002
TTD	0205	022D	0A2D

NOTE
SBT is represented in hex with the first two bits of the first character taken as '00' (only six bits are transmitted).

Table A–9 Control Characters (Hex)

Headers

A header is a sequence of characters beginning with SOH used for message type identification, message numbering, priority specification, and routing. Receipt of SOH initiates accumulation of BCC (but the SOH is not included in the BCC). The header can be a fixed or variable length, and ends with STX.

Text

The text part of the message contains the information to be sent between application programs. The text begins with STX and ends with ETX. Text can be broken up into blocks for better error control: each block ends with ETB. The normal end of a transmission is signalled by ETX followed by EOT. If a control character is detected, the receiving station terminates reception and waits for BCC.

Error Checking

BSC implements three types of error detection:

- VRC/LRC
- CRC–12
- CRC–16

VRC is an odd–parity check performed on each transmitted character, including the LRC characters at the end of the block. Each bit in the LRC character provides odd parity for the corresponding bit position of all characters sent in the block. In BSC, this character is called block check character and is sent as the next character following an ETB, ITB, or ETX character. The BCC sent with the data is compared at the receiver with one accumulated by the receiver, and if the two are equal, then the received message is valid. The BCC calculation is restarted by the first STX or SOH character received after the direction of transmission is reversed (called a line turnaround). All characters, except the SYN characters received from that point until the next line turnaround, are included in the calculation. If the message is sent in blocks with no line turnaround, the next block will be followed by ITB, then BCC. The BCC calculation is then restarted when the next STX or SOH character is received.

The CRC (cyclic redundancy check) codes are used for error checking in the same fashion as the LRC code. If the transmission code is set to SBT, use the CRC–12 method since each transmission character is only six bits. Always use the CRC–16 scheme for EBCDIC. For the ASCII code set, IBM suggests to use the VRC/LRC scheme in the non–transparency (standard) mode, and CRC–16 in transparency mode.

Transmission Code	Transparency Mode Not Installed	Transparency Mode Installed (Enabled)	Transparency Mode Installed (Disabled)
ASCII	VRC/LRC	CRC–16	VRC/CRC–16
EBCDIC	CRC–16	CRC–16	CRC–16
SBT	CRC–12	CRC–12	CRC–12

Table A–10 CRC Errors

A.4 Polling and Selection

The active participants on a BSC link are managed by a control station, which issues either a poll or a select message addressing the desired tributary. The poll sequence is an invitation-to-send from the control to the tributary. The tributary can then send any desired messages to the control station. The select is a request-to-receive notice from the control station, notifying the selected tributary station that a message will be sent from the control station.

Stations on the multidrop communication data link are assigned unique addresses which can consist of one to seven characters. The first character defines the station (cluster controller) and succeeding characters define some part of the station (device) such as the printer or display terminal, as required. IDACOM's implementation has a two character station address, where the character is repeated for increased reliability (see Appendix B).

A.5 Transparent–Text Mode

It is frequently necessary, when communicating between machines, to see data which does not represent characters, but instead some purely arbitrary quantity or object. One example is the binary representation of a computer program. In such a case, it is likely that some of the data has the same bit pattern as BSC control characters. Transparent-text mode, sometimes called transparency, allows data to be sent without being misinterpreted by the communications program. The basic technique in transparency is to precede each true control character with the DLE character. If a DLE bit pattern appears in the text of the message, it is preceded by a DLE. Thus, a bit pattern is interrupted as a control character only if preceded by a DLE. The resulting message, after processing by the link interface program to remove the DLE's, is shown in the following tables.

SYNC	DLE	STX	TRANS-PARENT TEXT	DLE	DLE	TRANS-PARENT TEXT	DLE	SYNC	TRANS-PARENT TEXT	DLE	ETB	BCC	SYNC	DLE	STX	TRANS-PARENT TEXT	DLE	ETX	BCC

Table A–11 Transmitted Block

NOTE
The first DLE in a DLE DLE sequence is stripped and the second is transmitted as part of the data.

SYNC	STX	TEXT	DLE	TEXT	SYNC	TEXT	ETB	BCC	STX	TEXT	ETX	BCC

Table A–12 Block after Stripping Transparency Control Character

NOTE
Transparent monitor mode or transparent mode is not supported in the BSC Monitor and Emulation applications.

A.6 Timeout Timers

Timeout timers are provided by the communication program (host system) or terminal to prevent delays caused by data errors or missing line turnaround signals. There are four timeout functions in BSC:

- Transmit
- Receive
- Disconnect
- Continue

Transmit timeout is normally set for one second, which defines the synchronous idle (SYN SYN or DLE SYN) sequences insertion rate in the data to help maintain bit and character synchronization.

Receive timeout is normally set for three seconds, and:

- limits the time a transmitting station waits for a reply;
- signals a receiving station to check the line for synchronous idle characters, indicating transmission is continuing. The receive timeout is reset each time a sync sequence is received; and
- sets a limit on the time a tributary station in a multipoint circuit can remain in control mode. The timer is reset each time an end signal such as an ENQ or ACK is received, as long as the station stays in control mode.

Disconnect timeout causes a station, on a switched network, to disconnect from the circuit after 20 seconds of inactivity.

Continue timeout causes a station transmitting a TTD to transmit another if still unable to send text. A receiving station must transmit a WACK if temporarily unable to receive within a two second interval.

A.7 Transmitting a Message

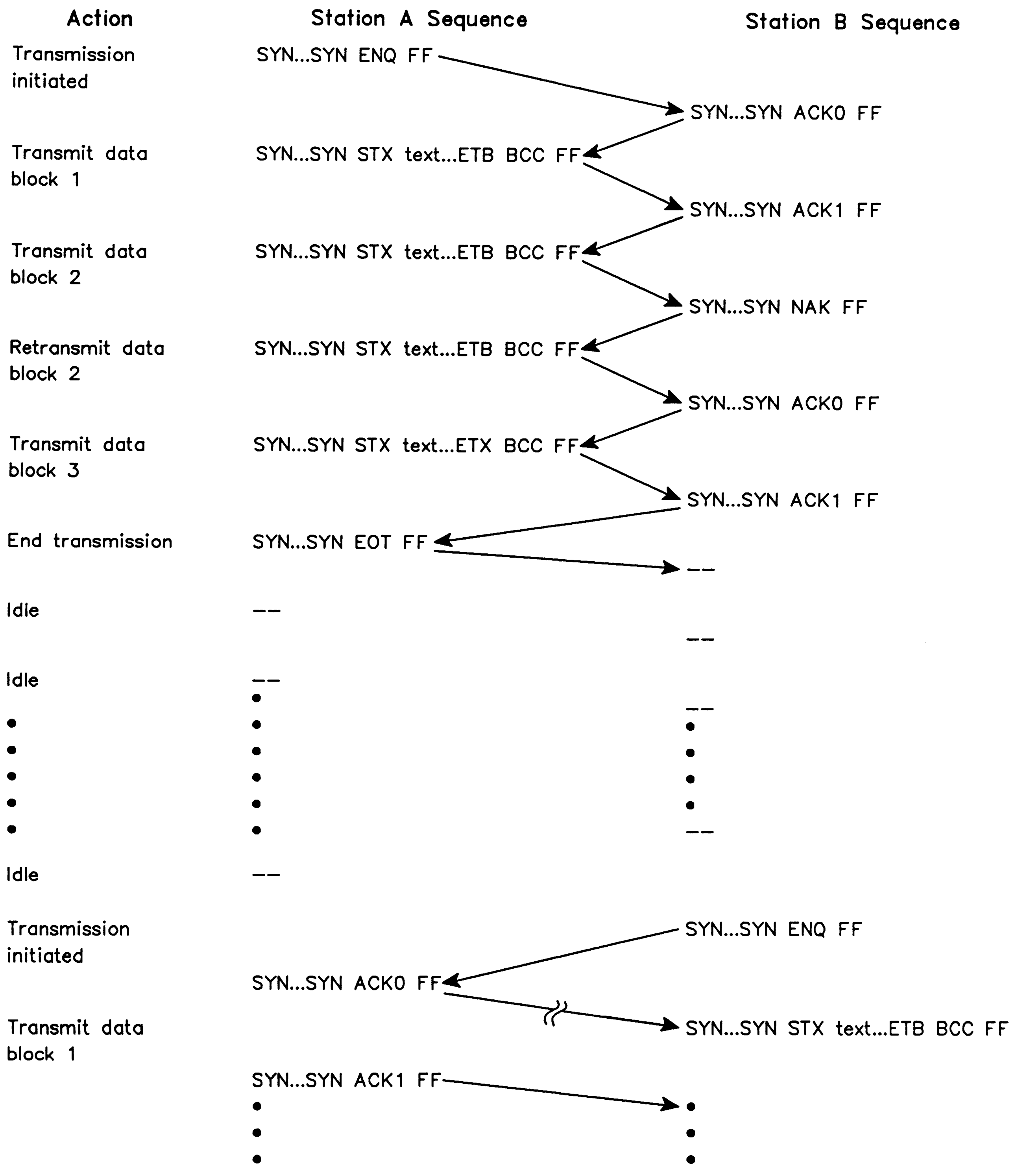


Figure A–3 Point–to–Point Setup and Transmission

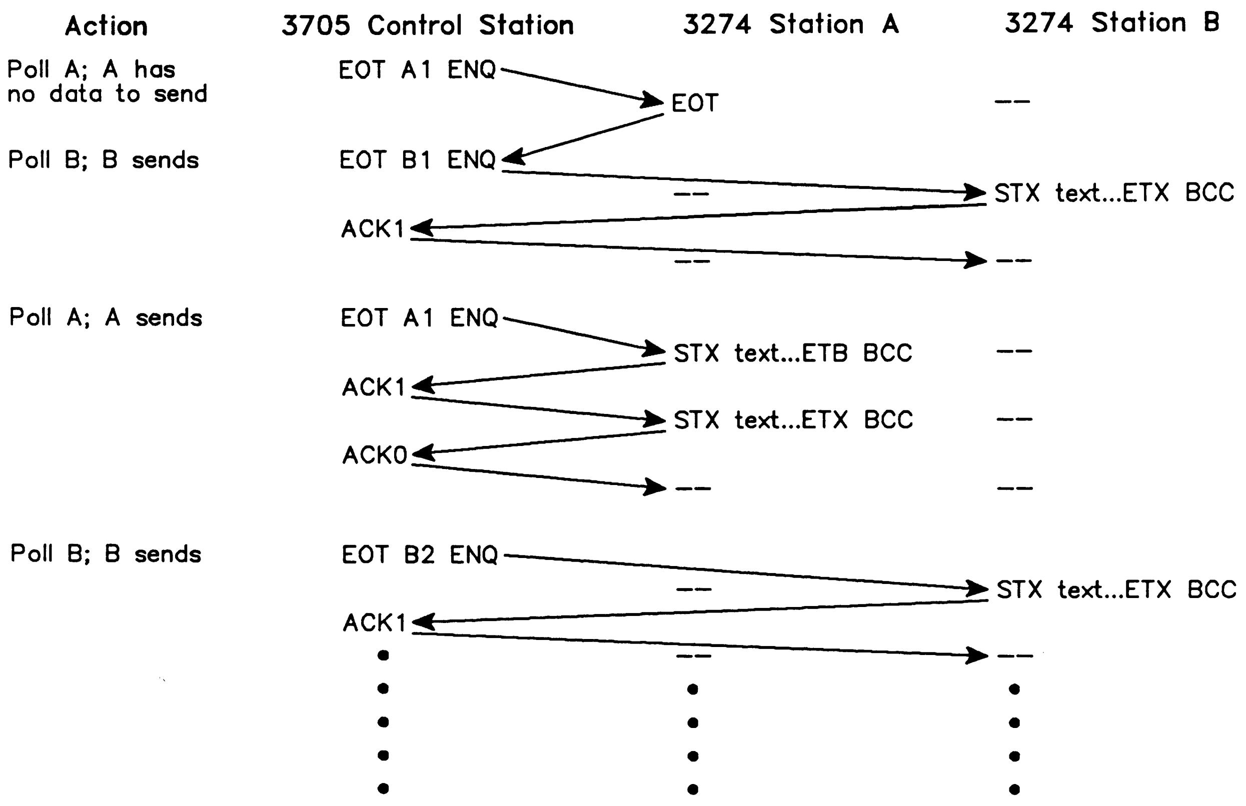


Figure A–4 Multipoint Polling Sequence Transmission

NOTE
To simplify the illustration, SYNC and PAD characters are not shown.

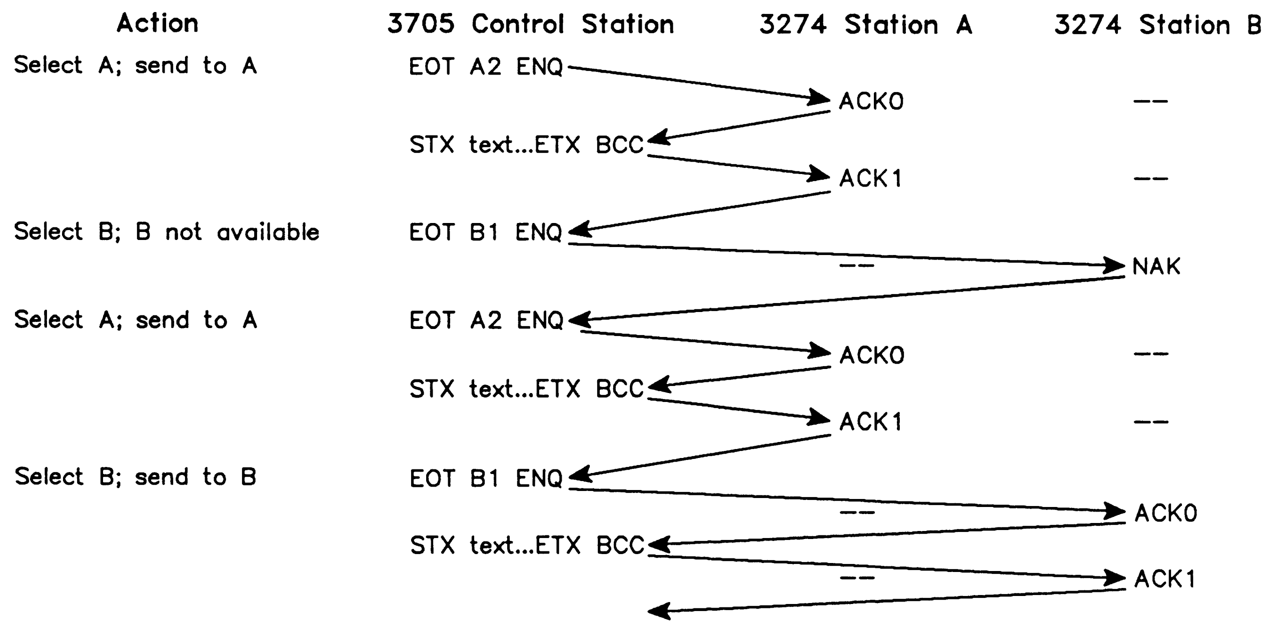


Figure A–5 Multipoint Select Sequence Transmission

NOTE
To simplify an illustration, SYN and PAD characters are not shown.

B
REMOTE CONTROL UNIT AND DEVICE ADDRESSING

Used for:

- Device selection
- Specific poll
- General poll
- Fixed return addresses

CU or Device Number	EBCDIC I/O Character	EBCDIC Hex (Note 1)	ASCII I/O Character	ASCII Hex
0	SP	40	SP	20
1	A	C1	A	41
2	B	C2	B	42
3	C	C3	C	43
4	D	C4	D	44
5	E	C5	E	45
6	F	C6	F	46
7	G	C7	G	47
8	H	C8	H	48
9	I	C9	I	49
10	cent	4A	[	5B
11	.	4B	.	2E
12	<	4C	<	3C
13	(	4D	(	28
14	+	4E	+	2B
15	| or !	4F	!	21
16	&	50	&	26
17	J	D1	J	4A
18	K	D2	K	4B
19	L	D3	L	4C
20	M	D4	M	4D
21	N	D5	N	4E
22	O	D6	O	4F
23	P	D7	P	50
24	Q	D8	Q	51
25	R	D9	R	52
26	!	5A	]	5D
27	$	5B	$	24
28	*	5C	*	2A
29	)	5D	)	29
30	;	5E	;	3B
31	¬ or ∧	5F	∧	5E

Used for:

- 3274 selection addresses
- Test requests

CU or Device Number	EBCDIC I/O Character	EBCDIC Hex (Note 1)	ASCII I/O Character	ASCII Hex
0	–	60	–	2D
1	/	61	/	2F
2	S	E2	S	53
3	T	E3	T	54
4	U	E4	U	55
5	V	E5	V	56
6	W	E6	W	57
7	X	E7	X	58
8	Y	E8	Y	59
9	Z	E9	Z	5A
10	|	6A	|	7C
11	,	6B	,	2C
12	%	6C	%	25
13	_	6D	_	5F
14	>	6E	>	3E
15	?	6F	?	3F
16	0	F0	0	30
17	1	F1	1	31
18	2	F2	2	32
19	3	F3	3	33
20	4	F4	4	34
21	5	F5	5	35
22	6	F6	6	36
23	7	F7	7	37
24	8	F8	8	38
25	9	F9	9	39
26	:	7A	:	3A
27	#	7B	#	23
28	@	7C	@	40
29	'	7D	'	27
30	=	7E	=	3D
31	"(Note 2)	7F	"	22

3274 Addresses		EBCDIC	ASCII
General poll CU5	CU Address	C5 C5	45 45
	Device Address	7F 7F	22 22
Specific poll device 4 on CU5	CU Address	C5 C5	45 45
	Device Address	C4 C4	44 44
Select device 4 on CU5	CU Address	E5 E5	56 56
	Device Address	C4 C4	44 44

Table B–1 Remote Control Unit and Device Addressing

NOTE

1. *Graphic characters for the United States I/O interface codes are shown. Graphic characters for EBCDIC 4A, 5A, 5B, 7B, 7C, and 7F might differ for particular world Trade I/O interface codes. Refer to IBM 3270 Information Display System: Character Set Reference, GA27–2837, for possible graphic differences when these codes are used.*
2. *I/O character address (") is used as the device address to specify a General Poll operation.*

C
POINTERS AND LENGTHS

The following diagrams display the breakdown of various types of messages using the L2-POINTER, L3-POINTER, L4-POINTER, L2-LENGTH, L3-LENGTH, and L4-LENGTH variables. Refer to Section 8.1 for descriptions of these variables.

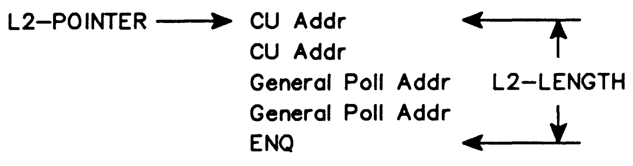


Figure C-1 General Poll Sequence

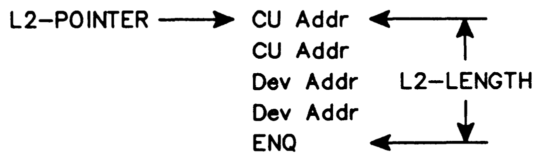


Figure C-2 Specific Poll Sequence

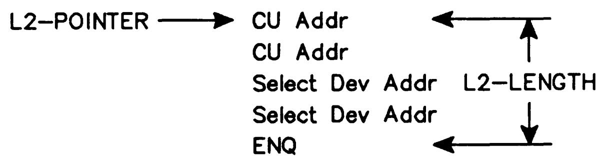


Figure C-3 Select Sequence

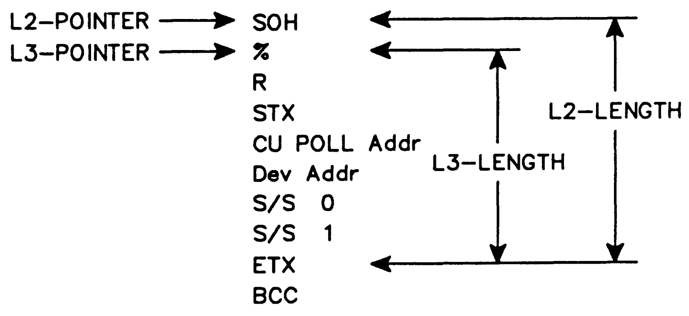


Figure C-4 Status Message

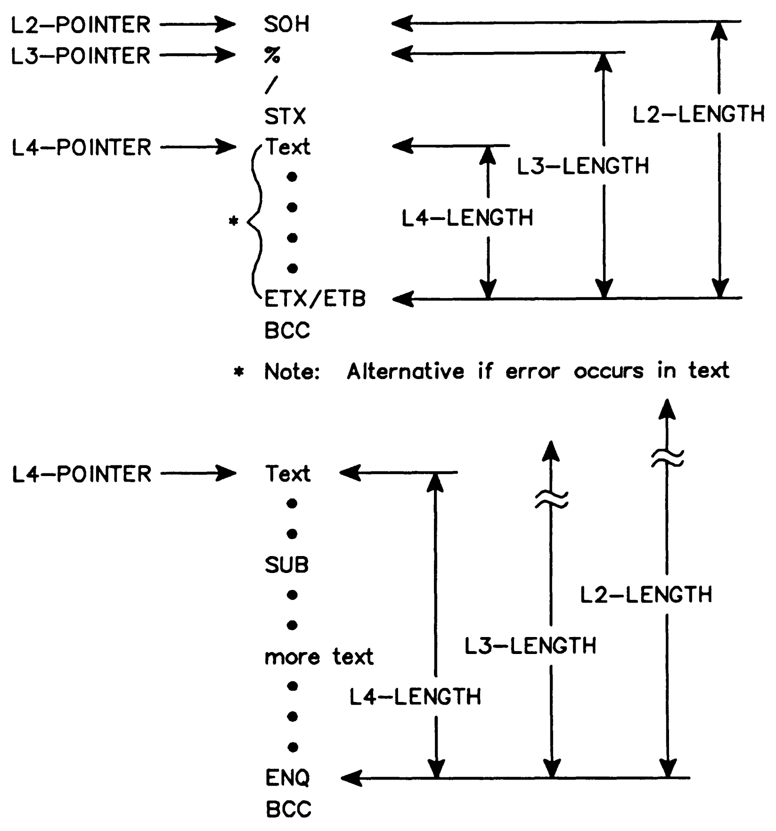


Figure C-5 Test Request Message

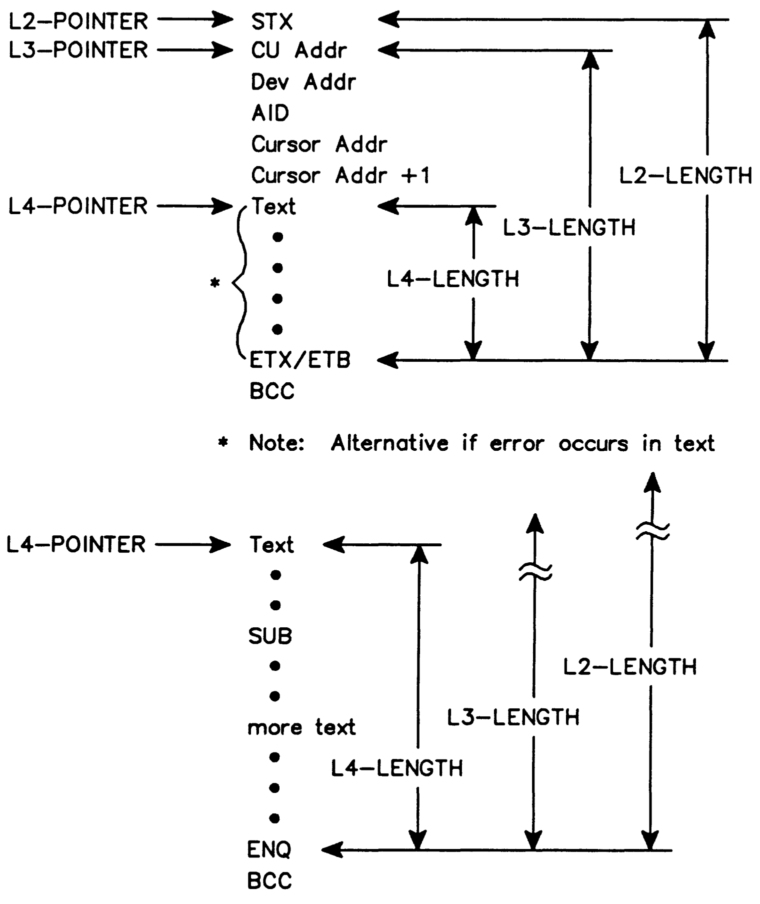


Figure C-6 Read Modified/Short Read

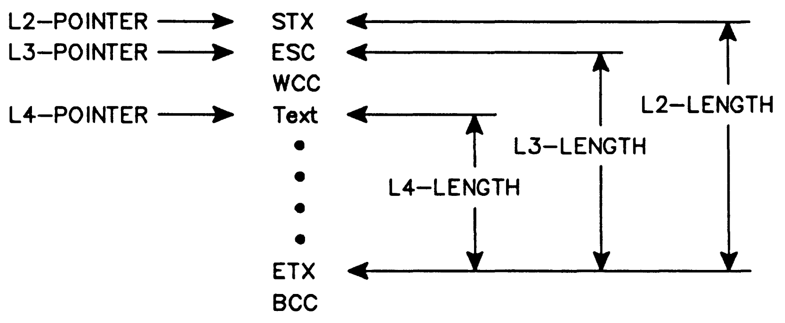


Figure C-7 Write Command

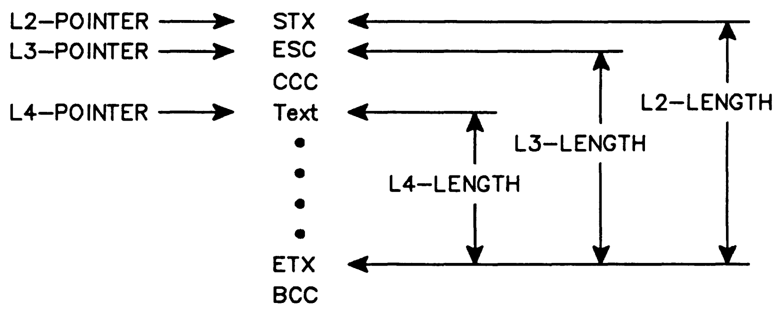


Figure C-8 Copy Command

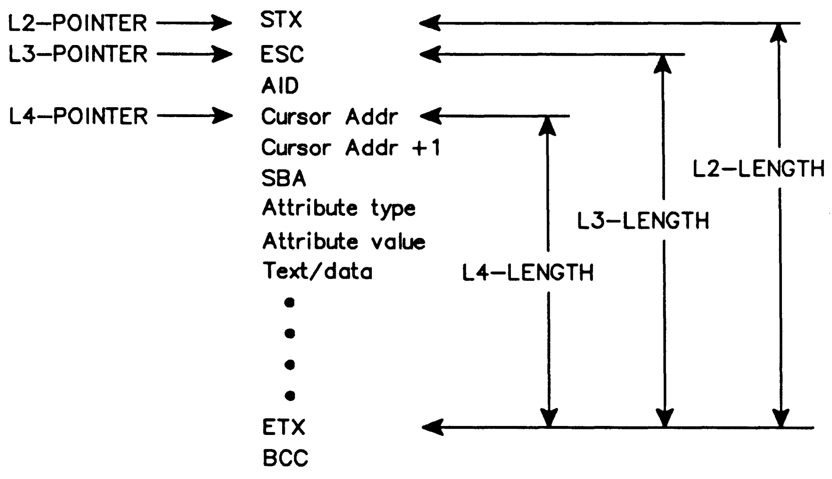


Figure C-9 Read Buffer Command

D
CODING CONVENTIONS

The following section outlines some coding and style conventions recommended by IDACOM. Although you can develop your own style, it is suggested to stay close to these standards to enhance readability.

D.1 Stack Comments

A stack comment is surrounded by parentheses, and shows two stack pictures. The first picture shows any items or 'input parameters' that are consumed by the command; the second picture shows any items or 'output parameters' returned by the command.

Example:
The '=' command has the following stack comment.

(n_1\\n_2 -- flag)

In this example, n_1 and n_2 are numbers and the flag is either 0 for a false result, or 1 for a true result. This same example could also be written as follows.

(n_1\\n_2 -- 0|1)

The '\' character separates parameters when there is more than one. The parameters are listed from left to right with the leftmost item representing the bottom of the stack and the rightmost item representing the top of the stack.

The '|' character indicates that there is more than one possible output. The above example indicates that either a 0 or a 1 is returned on the stack after the '=' operation, with 0 being a false result, and 1 a true result.

D.2 Stack Comment Abbreviations

Following is a list of commonly used abbreviations. In most cases the stack comments shown in this manual have been written in full rather than abbreviated.

Symbol	Description
a	Memory address
b	8 bit byte
c	7 bit ASCII character
n	16 bit signed integer
d	32 bit signed integer
u	32 bit unsigned integer
f	Boolean flag (0=false, non-zero=true)
ff	Boolean false flag (zero)
tf	Boolean true flag (non-zero)
s	String (actual address of a character string which is stored in a count prefixed manner)

Table D-1 ITL Symbols

D.3 Program Comments

Program comments appear in source code surrounded by parentheses. These describe the intent or purpose of the definition or line of code.

There must be at least one space on each side of the parentheses.

Example:

```
:  HELLO  ( -- )                        ( Display text Hello in Notice Window )
      " HELLO"                          ( Create string )
      W.NOTICE                          ( Output to Notice Window )
;
```

The program comment should be kept to a minimum and yet contain enough information that another programmer can tell the intent at a glance.

D.4 Test Manager Constructs

Coding conventions for user test scripts should generally follow the style presented throughout this manual.

Indenting nested program structures should be done using the tab key in the editor. Furthermore, the use of many meaningful comments is highly recommended and will enhance the continued maintainability of the program.

Example:
(State definition purpose comment)

```
0 STATE{
      EVENT Recognition Commands          ( Comment )
      ACTION{
          Action Commands                 ( Comment )
          IF
              ...                         ( Comment )
              ...                         ( Comment )
          ENDIF
      }ACTION
  }STATE
```

D.5 Spacing and Indentation Guidelines

The following list outlines the general guidelines for spacing and indentations:

- One space between colon and name in colon definitions.
- One space between opening parenthesis and text in comments.
- One space between numbers and words within a definition.
- One space between initial " in strings (i.e. with " string", W." string", T." string", P." string", X" hex characters", etc...)
- One or more spaces at the end of each line unless defining string which requires additional characters.
- Tab for nested constructs.
- Carriage return after colon definition and stack comment.
- Carriage return after last line of code in colon definition and semi-colon.

See the examples in Appendices D.4 and D.6.

D.6 Colon Definitions

Colon definition should be preceded by a short comment. The colon definition should start at the first column of a line. All code underneath the definition name should be preceded by one tab. Each element within the colon definition should be well defined.

Example:
(Description of command)

```
:  COMMANDNAME                              ( Stack description )
      .....                                 ( Comment for first line of code )
      IF
          ....                              ( Comment )
          DOCASE
              CASE  X  [ ... ]              ( Comment )
              CASE  Y  [ ... ]              ( Comment )
              CASE  DUP  [ ... ]            ( Comment )
          ENDCASE
      ELSE
          BEGIN
              .....                         ( Comment )
              .....                         ( Comment )
          UNTIL
      ENDIF
;
```

E
ASCII/EBCDIC/HEX CONVERSION TABLE

HEX	DEC	OCT	ASCII	EBCDIC
00	0	00	NUL	NUL
01	1	01	SOH	SOH
02	2	02	STX	STX
03	3	03	ETX	ETX
04	4	04	EOT	PF
05	5	05	ENQ	HT
06	6	06	ACK	LC
07	7	07	BEL	DEL
08	8	10	BS	GE
09	9	11	HT	SPS
0A	10	12	LF	RPT
0B	11	13	VT	VT
0C	12	14	FF	FF
0D	13	15	CR	CR
0E	14	16	SO	SO
0F	15	17	SI	SI
10	16	20	DLE	DLE
11	17	21	DC1	DC1
12	18	22	DC2	DC2
13	19	23	DC3	DC3
14	20	24	DC4	RES
15	21	25	NAK	NL
16	22	26	SYN	BS
17	23	27	ETB	POC
18	24	30	CAN	CAN
19	25	31	EM	EM
1A	26	32	SUB	UBS
1B	27	33	ESC	CUI
1C	28	34	FS	IFS
1D	29	35	GS	IGS
1E	30	36	RS	IRS
1F	31	37	US	IUS
20	32	40	SP	DS
21	33	41	!	SOS
22	34	42	″	FS
23	35	43	#	WUS
24	36	44	$	BYP
25	37	45	%	LF
26	38	46	&	ETB
27	39	47	′	ESC
28	40	50	(	SA
29	41	51	)	SFE
2A	42	52	*	SM/SW
2B	43	53	+	CSP
2C	44	54	,	MFA
2D	45	55	–	ENQ
2E	46	56	.	ACK
2F	47	57	/	BEL
30	48	60	0	
31	49	61	1	
32	50	62	2	SYN
33	51	63	3	IR
34	52	64	4	PP
35	53	65	5	TRN
36	54	66	6	NBS
37	55	67	7	EOT
38	56	70	8	SBS
39	57	71	9	IT
3A	58	72	:	RFF
3B	59	73	;	CU3
3C	60	74	<	DC4
3D	61	75	=	NAK
3E	62	76	>	
3F	63	77	?	SUB
40	64	100	@	SP
41	65	101	A	
42	66	102	B	
43	67	103	C	
44	68	104	D	
45	69	105	E	
46	70	106	F	
47	71	107	G	
48	72	110	H	
49	73	111	I	
4A	74	112	J	cent
4B	75	113	K	.
4C	76	114	L	<
4D	77	115	M	(
4E	78	116	N	+
4F	79	117	O	|
50	80	120	P	&
51	81	121	Q	
52	82	122	R	
53	83	123	S	
54	84	124	T	
55	85	125	U	
56	86	126	V	
57	87	127	W	
58	88	130	X	
59	89	131	Y	
5A	90	132	Z	!
5B	91	133	[	$
5C	92	134	\	*
5D	93	135	]	)
5E	94	136	ˆ	;
5F	95	137	_	¬

HEX	DEC	OCT	ASCII	EBCDIC
60	96	140	`	−
61	97	141	a	/
62	98	142	b	
63	99	143	c	
64	100	144	d	
65	101	145	e	
66	102	146	f	
67	103	147	g	
68	104	150	h	
69	105	151	i	
6A	106	152	j	|
6B	107	153	k	,
6C	108	154	l	%
6D	109	155	m	_
6E	110	156	n	>
6F	111	157	o	?
70	112	160	p	
71	113	161	q	^
72	114	162	r	
73	115	163	s	
74	116	164	t	
75	117	165	u	
76	118	166	v	
77	119	167	w	
78	120	170	x	
79	121	171	y	\
7A	122	172	z	:
7B	123	173	{	#
7C	124	174	|	@
7D	125	175	}	'
7E	126	176	˜	=
7F	127	177	DEL	"
80	128	200		
81	129	201		a
82	130	202		b
83	131	203		c
84	132	204		d
85	133	205		e
86	134	206		f
87	135	207		g
88	136	210		h
89	137	211		i
8A	138	212		
8B	139	213		{
8C	140	214		≤
8D	141	215		(
8E	142	216		+
8F	143	217		†

HEX	DEC	OCT	ASCII	EBCDIC
90	144	220		
91	145	221		j
92	146	222		k
93	147	223		l
94	148	224		m
95	149	225		n
96	150	226		o
97	151	227		p
98	152	230		q
99	153	231		r
9A	154	232		
9B	155	233		}
9C	156	234		□
9D	157	235		)
9E	158	236		±
9F	159	237		■
A0	160	240		−
A1	161	241		o
A2	162	242		s
A3	163	243		t
A4	164	244		u
A5	165	245		v
A6	166	246		w
A7	167	247		x
A8	168	250		y
A9	169	251		z
AA	170	252		
AB	171	253		└
AC	172	254		┌
AD	173	255		[
AE	174	256		≥
AF	175	257		•
B0	176	260		0
B1	177	261		1
B2	178	262		2
B3	179	263		3
B4	180	264		4
B5	181	265		5
B6	182	266		6
B7	183	267		7
B8	184	270		8
B9	185	271		9
BA	186	272		
BB	187	273		┘
BC	188	274		┐
BD	189	275		]
BE	190	276		≠
BF	191	277		−

HEX	DEC	OCT	ASCII	EBCDIC
C0	192	300		{
C1	193	301		A
C2	194	302		B
C3	195	303		C
C4	196	304		D
C5	197	305		E
C6	198	306		F
C7	199	307		G
C8	200	310		H
C9	201	311		I
CA	202	312		
CB	203	313		
CC	204	314		
CD	205	315		
CE	206	316		
CF	207	317		
D0	208	320		}
D1	209	321		J
D2	210	322		K
D3	211	323		L
D4	212	324		M
D5	213	325		N
D6	214	326		O
D7	215	327		P
D8	216	330		Q
D9	217	331		R
DA	218	332		
DB	219	333		
DC	220	334		
DD	221	335		
DE	222	336		
DF	223	337		
E0	224	340		\
E1	225	341		
E2	226	342		S
E3	227	343		T
E4	228	344		U
E5	229	345		V
E6	230	346		W
E7	231	347		X
E8	232	350		Y
E9	233	351		Z
EA	234	352		
EB	235	353		
EC	236	354		
ED	237	355		
EE	238	356		
EF	239	357		

HEX	DEC	OCT	ASCII	EBCDIC
F0	240	360		0
F1	241	361		1
F2	242	362		2
F4	244	364		4
F3	243	363		3
F5	245	365		5
F6	246	366		6
F7	247	367		7
F8	248	370		8
F9	249	371		9
FA	250	372		
FB	251	373		
FC	252	374		
FD	253	375		
FE	254	376		
FF	255	377		

F
COMMAND CROSS REFERENCE LIST

This appendix cross references old commands and variables, not appearing in this manual, with new replacement commands. Refer to previous versions of this manual for descriptions of old commands. The new commands achieve the same function; however, the input/output parameters may have changed.

Old Command	New Command
CCU= n	n C-SEL-CU ! C+CU
CLU= n	n C-SEL-LU ! C+LU
CLUST_LU cu	-1 cu ACT
DATA-STATUS	STATUS_ERR?
DCU= n	n D-SEL-CU ! D+CU
DEV lu	-1 lu cu ACT
DLU= n	n D-SEL-LU ! D+LU
PLAY-COUNT	BLOCK-COUNT
PLAY-ETIME	END-TIME
PLAY-ID	PORT-ID
PLAY-STIME	START-TIME
RCU= n	n R-SEL-CU ! R+CU
REC-STATUS	STATUS_ERR?
RLU= n	n R-SEL-LU ! R+LU
SET_LONG n	n LONG INTERVAL !
SET_SHORT n	n SHORT-INTERVAL !
SET_SPEED n	n INTERFACE-SPEED !
START_POLLING	POLL_ON
STOP_POLLING	POLL_OFF

INDEX

INDEX [continued]

INDEX [continued]

INDEX [continued]

INDEX [continued]